Logical Methods in Computer Science
Volume 19, Issue 3, 2023, pp. 12:1–12:49
https://lmcs.episciences.org/

Submitted Oct. 29, 2021
Published Aug. 09, 2023

THE COMPLEXITY OF AGGREGATES OVER EXTRACTIONS BY REGULAR EXPRESSIONS *

JOHANNES DOLESCHAL [a,c], BENNY KIMELFELD [b], AND WIM MARTENS [a]

[a] University of Bayreuth, Germany
e-mail address: johannes.doleschal@uni-bayreuth.de, wim.martens@uni-bayreuth.de

[b] Technion, Haifa, Israel
e-mail address: bennyk@cs.technion.ac.il

[c] Hasselt University, Belgium

Abstract. Regular expressions with capture variables, also known as "regex-formulas," extract relations of spans (intervals identified by their start and end indices) from text. In turn, the class of regular document spanners is the closure of the regex formulas under the Relational Algebra. We investigate the computational complexity of querying text by aggregate functions, such as sum, average, and quantile, on top of regular document spanners. To this end, we formally define aggregate functions over regular document spanners and analyze the computational complexity of exact and approximate computation. More precisely, we show that in a restricted case, all studied aggregate functions can be computed in polynomial time. In general, however, even though exact computation is intractable, some aggregates can still be approximated with fully polynomial-time randomized approximation schemes (FPRAS).

1. Introduction

Information extraction commonly refers to the task of extracting structured information from text. A document spanner (or just spanner for short) is an abstraction of an information extraction program: it states how to transform a document into a relation over its spans. More formally, a *document* is a string d over a finite alphabet, a *span* of d represents a substring of d by its start and end positions, and a *spanner* is a function that maps every document d into a relation over the spans of d [FKRV15a]. The spanner framework has originally been introduced as the theoretical basis underlying IBM's SQL-like rule system for information extraction, namely SystemT [KLR$^+$08, LRC11]. The most studied spanner instantiation is the class of *regular spanners*, which is the closure of regex formulas (regular expressions with capture variables) under the standard operations of the relational algebra (projection, natural join, union, and difference). Equivalently, the regular spanners are the ones expressible as *variable-set automata* (VSet-automata for short), which are nondeterministic finite-state

Key words and phrases: Information extraction, document spanners, weighted automata, aggregation, annotated databases, provenance semirings.

* A short version of this article has been published in a conference proceedings [DBKM21].

DOI:10.46298/LMCS-19(3:12)2023

automata that can open and close capture variables. These spanners extract from the text relations wherein the capture variables are the attributes.

While regular spanners and natural generalizations thereof are the basis of rule-based systems for text analytics, they are also used implicitly in other types of systems, and particularly ones based on statistical models and machine learning. Rules similar to regular spanners are used for *feature generators* of graphical models (e.g., Conditional Random Fields) [LBC04, SM12], *weak constraints* of Markov Logic Networks [PD07] and extensions such as DeepDive [SWW$^+$15], and the generators of *noisy training data* ("labeling functions") in the state-of-the-art Snorkel system [RBE$^+$17]. Further connections to regular spanners can potentially arise from efforts to express artificial neural networks for natural language processing as finite-state automata [MY18, MSV$^+$19, WGY18]. The computational complexity of evaluating regular spanners has been well studied from various angles, including the data and combined complexity of answer enumeration [ABMN19, FRU$^+$18, FKP18, MRV18], the cost of combining spanners via relational algebra operators [PFKK19] and recursive programs [PtCFK19], their dynamic complexity [FT20], evaluation in the presence of weighted transitions [DKMP22], and the ability to distribute their evaluation over fragments of the document [DKM$^+$19].

In this article, we study the computational complexity of evaluating *aggregate functions* over regular spanners. These are queries that map a document d and a spanner S into a number $\alpha(S(d))$, where $S(d)$ is the relation obtained by applying S to d and α is a standard aggregate function: Count, Sum, Average, Min, Max, or Quantile. There are various scenarios where queries that involve aggregate functions over spanners can be useful. For example, such queries arise in the extraction of statistics from textual resources like medical publications [NKS$^+$19] and news reports [SC09]. As another example, when applying advanced text search or protein/DNA motif matching using regular expressions [CG89, NG94], the search engine typically provides the (exact or approximate) number of answers, and we would like to be able to compute this number without actually computing the answers, especially when the number of answers is prohibitively large. Finally, when programming feature generators or labeling functions in extractor development, the programmer is likely to be interested in aggregate statistics and summaries for the extractions (e.g., to get a holistic view of what is being extracted from the dataset, such as quantiles over extracted ages and so on), and again, we would like to be able to estimate these statistics faster than it takes to materialize the entire set of answers.

Our main objective in this work is to understand when it is tractable to compute $\alpha(S(d))$. This question raises closely related questions that we will discuss, such as when the materialization of the set of intermediate results $S(d)$ (which can be exponentially large) can be avoided. Furthermore, when the exact computation of $\alpha(S(d))$ is intractable, we study whether it can be approximated.

At the technical level, each aggregate function (with the exception of Count) requires a specification of how an extracted tuple of spans represents a number. For example, the number 21 can be represented by the span of the string "21", "21.0", "twenty one", "twenty first", "three packs of seven" and so on. To abstract away from specific textual representations of numbers, we consider several means of assigning weights to tuples. To this end, we assume that a (representation of a) *weight function* w, which maps every tuple of $S(d)$ into a number, is part of the input of the aggregate functions. Hence, the general form of the aggregate query we study is $\alpha(S, d, w)$. The direct approach to evaluating $\alpha(S, d, w)$ is to compute $S(d)$, apply w to each tuple, and apply α to the resulting sequence of numbers. This approach

works well if the number of tuples in $S(d)$ is manageable (e.g., bounded by some polynomial). However, the number of tuples in $S(d)$ can be exponential in the number of variables of S, and so, the direct approach takes exponential time in the worst case. We will identify several cases in which $S(d)$ is exponential, yet $\alpha(S(d))$ can be computed in polynomial time.

It is not very surprising that, at the level of generality we adopt, each of the aggregate functions is intractable (#P-hard) in general. Hence, we focus on several assumptions that can potentially reduce the inherent hardness of evaluation:

- Restricting the range of weight functions to positive numbers;
- Restricting to weight functions that are determined by a single span or defined by (unambiguous) weighted VSet-automata;
- Restricting to spanners that are represented by an unambiguous variant of VSet-automata;
- Allowing for a randomized approximation (FPRAS, i.e., fully polynomial randomized approximation schemes).

Our analysis shows which of these assumptions brings the complexity down to polynomial time, and which is insufficient for tractability. Importantly, we derive an interesting and general tractable case for each of the aggregate functions we study.

To the best of our knowledge, counting the number of tuples extracted by a VSet-automaton (i.e., the Count aggregate function) is the only aggregation function for document spanners which has been studied in literature, except for Doleschal et al. [DKMP22] who consider a variation of maximum aggregation. (Given a weighted VSet-automaton and a document, they study the computational complexity of returning a tuple with maximal weight.) Concerning counting, Florenzano et al. [FRU$^+$18] study the problem of counting the number of extractions of a VSet-automaton and approximation thereof is studied by Arenas et al. [ACJR19]. To be specific, Arenas et al. [ACJR19] give a polynomial-time uniform sampling algorithm from the space of words which are accepted by an NFA and have a given length. Using that sampling, they establish an FPRAS for the Count aggregate function. Our FPRAS results are based on their results. We explain the connection between the known results and our work in more detail throughout the article. Yet, to the best of our knowledge, this work is the first to consider aggregate functions over numerical values extracted by document spanners.

Comparison to the Conference Version. Compared to the conference version of this article [DBKM21], the following aspects are new. We now consider constant-width weight functions, which generalize the single-variable weight functions from [DBKM21]; Section 4.2 is new; and we provide a more detailed complexity overview for regular weight functions over different semirings. Furthermore, proofs that were missing in [DBKM21] are now included. On a technical level, we now use parsimonious reductions and metric reductions instead of Turing reductions for some of the results, which strengthens them.

Organization. This article is organized as follows. In Section 2, we give preliminary definitions and notation. We summarize the main results in Section 3 and expand on these results in the later sections. In Section 4 we give some preliminary results. We describe our investigation for constant-width weight functions, polynomial-time weight functions and regular weight functions in Sections 5, 6 and 7, respectively. Finally, we study approximation in Section 8 and conclude in Section 9.

2. Preliminaries

We define here the main concepts and notation that we will use throughout the article.

2.1. **Sets, Multisets, and Semirings.** The cardinality of a set A is denoted by $|A|$. A *multiset* over A is a function $M : A \to \mathbb{N}$. We call $M(a)$ the *multiplicity* of a in M and say that $a \in M$ if $M(a) > 0$. The *size* of M denoted $|M|$, is the sum of the multiplicities of all elements in A, that is $\sum_{a \in A} M(a)$. Note that $|M|$ may be infinite. We denote multisets in brackets $\{\!\{\cdot\}\!\}$ in the usual way. For example, in the multiset $M = \{\!\{1, 1, 3\}\!\}$ we have $M(1) = 2$, $M(3) = 1$, and $|M| = 3$. Furthermore, given a set X, we denote by 2^X the powerset of X, i.e., the set of all subsets of X.

A *commutative monoid* $(\mathbb{M}, *, \mathrm{id})$ is an algebraic structure consisting of a set $\mathbb{M}$, a binary operation $*$ and an element $\mathrm{id} \in \mathbb{M}$, such that:

(1) $*$ is associative, i.e., $(a * b) * c = a * (b * c)$ for all $a, b, c \in \mathbb{M}$,
(2) $*$ is commutative, i.e. $a * b = b * a$ for all $a, b \in \mathbb{M}$, and
(3) id is an identity, i.e., $\mathrm{id} * a = a$ for all $a \in \mathbb{M}$.

A *commutative semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is an algebraic structure consisting of a set $\mathbb{K}$, containing two elements: the *zero* element $\overline{0}$ and the *one* element $\overline{1}$. Furthermore, it is equipped with two binary operations, namely *addition* $\oplus$ and *multiplication* $\otimes$ such that:

(1) $(\mathbb{K}, \oplus, \overline{0})$ and $(\mathbb{K}, \otimes, \overline{1})$ are commutative monoids,
(2) multiplication distributes over addition, that is, $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$, for all $a, b, c \in \mathbb{K}$, and
(3) $\overline{0}$ is absorbing for $\otimes$, that is, $\overline{0} \otimes a = \overline{0}$ for all $a \in \mathbb{K}$.

Example 2.1. The following are commutative semirings.

(1) The *numeric semiring* $(\mathbb{Q}, +, \cdot, 0, 1)$ over the rationals, with the usual addition and multiplication operators.
(2) The *Boolean semiring* $(\mathbb{B}, \vee, \wedge, \mathsf{false}, \mathsf{true})$ where $\mathbb{B} := \{\mathsf{false}, \mathsf{true}\}$.
(3) The *tropical semiring* $(\mathbb{T}, \min, +, \infty, 0)$ where $\mathbb{T} := \mathbb{Q} \cup \{\infty\}$ and min stands for the binary minimum function. □

2.2. **Document Spanners and $\mathbb{K}$-Annotators.** This article is within the formalism of *document spanners* by Fagin et al. [FKRV15a, FKRV15b]. More precisely, we use the notion of $\mathbb{K}$-annotators, as introduced by Doleschal et al. [DKMP22], which enables document spanners to annotate provenance information. Next, we revisit the definitions of these concepts. We assume countably infinite and disjoint sets D and Vars, containing *data values* (or simply *values*) and *variables*, respectively.

Documents and Spans. Let Γ be a finite set, disjoint from D and Vars, of symbols. We refer to Γ as an *alphabet*. A sequence $s = \sigma_1 \cdots \sigma_n$ of symbols where every $\sigma_i \in \Gamma$ is a *string* over the set Γ. If $n = 0$ we denote s by ε and call s *empty*. By Γ^* we denote the set of all strings over Γ. We denote by $|s|$ the length n of a string $s \in \Gamma^*$. In the context of Information Extraction, we will restrict ourselves to a subset of symbols of Γ, which we will always denote as Σ. We typically use the letter d (and indexed variations thereof) to denote strings over Σ and refer to such strings as *documents*.

```
T h e r e ␣ a r e ␣ 7 ␣ e v e n t s ␣ i n ␣ B e l g i u m , ␣ 1 0 - 1 5 ␣ i n ␣
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
F r a n c e , ␣ 4 ␣ i n ␣ L u x e m b o u r g , ␣ t h r e e ␣ i n ␣ B e r l i n .
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
```

x_{loc}	x_{events}
$[23, 30\rangle$	$[11, 12\rangle$
$[41, 47\rangle$	$[32, 37\rangle$
$[54, 64\rangle$	$[49, 50\rangle$
$[75, 81\rangle$	$[66, 71\rangle$

$d_{x_{\mathsf{loc}}}$	$d_{x_{\mathsf{events}}}$
Belgium	7
France	10-15
Luxembourg	4
Berlin	three

Figure 1: A document d (top), a span relation R (bottom left) and the corresponding string relation (bottom right).

A *span* of d is an expression of the form $[i, j\rangle$ with $1 \leq i \leq j \leq n + 1$. For a span $[i, j\rangle$ of d, we denote by $d_{[i,j\rangle}$ the string $\sigma_i \cdots \sigma_{j-1}$. A span $[i, j\rangle$ is empty if $i = j$ which implies that $d_{[i,j\rangle} = \varepsilon$. Two spans $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$ are *equal* if $i_1 = i_2$ and $j_1 = j_2$. In particular, we observe that two spans do not have to be equal if they select the same string. That is, $d_{[i_1,j_1\rangle} = d_{[i_2,j_2\rangle}$ does not imply that $[i_1, j_1\rangle = [i_2, j_2\rangle$. For a document d, we denote by $\mathsf{Spans}(d)$ the set of all possible spans of d and by Spans the set of all possible spans of all possible documents.

$\mathbb{K}$*-relations and* $\mathbb{K}$*-annotators.* Let $V \subseteq \mathsf{Vars}$ be a finite set of variables. A V*-tuple* is a function $\mathrm{t} : V \to \mathsf{D}$ that assigns values to variables in V. We sometimes leave V implicit when the precise set is not important. For such a tuple t, we denote the set V by $\mathsf{Vars}(\mathrm{t})$. We denote the set of all V-tuples by V-Tup. For a subset $X \subseteq \mathsf{Vars}$, we denote the restriction of t to the variables in X by $\pi_X(\mathrm{t})$ or simply $\pi_X\mathrm{t}$. We say that a tuple t is *empty*, denoted by $\mathrm{t} = ()$, if $\mathsf{Vars}(\mathrm{t}) = \emptyset$.

A $\mathbb{K}$*-relation* R *over* V is a function $R : V\text{-}\mathsf{Tup} \to \mathbb{K}$ such that its *support*, defined by $\mathsf{Supp}(R) := \{\mathrm{t} \mid R(\mathrm{t}) \neq \overline{0}\}$, is finite. We will also write $\mathrm{t} \in R$ to abbreviate $\mathrm{t} \in \mathsf{Supp}(R)$. Furthermore, we say that two $\mathbb{K}$-relations R_1 and R_2 are *disjoint* if $\mathsf{Supp}(R_1) \cap \mathsf{Supp}(R_2) = \emptyset$. The *size* of a $\mathbb{K}$-relation R is the size of its support, that is, $|R| := |\mathsf{Supp}(R)|$. The *arity* of a V-tuple t is the cardinality $|V|$ of V and, similarly, the *arity* of a $\mathbb{K}$-relation over V is $|V|$.

The framework focuses on functions that extract spans from documents and assigns them to variables. Since we will be working with relations over spans, also called *span relations*, we assume that D is such that $\mathsf{Spans} \subseteq \mathsf{D}$. A *d-tuple* t is a V-tuple which only assigns values from $\mathsf{Spans}(d)$, that is, $\mathrm{t}(x) \subseteq \mathsf{Spans}(d)$ for every $x \in \mathsf{Vars}(\mathrm{t})$. If the document d is clear from the context, we sometimes say simply *tuple* instead of d-tuple. We denote by d_{t} the tuple $(d_{\mathrm{t}(x_1)}, \ldots, d_{\mathrm{t}(x_n)})$, where $\mathsf{Vars}(\mathrm{t}) = \{x_1, \ldots, x_n\}$.

A $\mathbb{K}$*-weighted span relation over document* d *and variables* V is a $\mathbb{K}$-relation R wherein every tuple is a d-tuple t with $\mathsf{Vars}(\mathrm{t}) = V$. We also denote V by $\mathsf{Vars}(R)$. A $\mathbb{K}$*-weighted string relation* is a $\mathbb{K}$-relation R wherein every tuple $\mathrm{t} \in R$ assigns strings, that is, $\mathrm{t}(x) \in \Sigma^*$ for every variable $x \in \mathsf{Vars}(\mathrm{t})$. Note that we can associate a string relation to every span relation over a document d by replacing every span $[i, j\rangle$ with the string $d_{[i,j\rangle}$.

Example 2.2. Consider the document in Figure 1. The table on the bottom left depicts a ($\mathbb{B}$-weighted) span relation R, encoding a possible extraction of locations with the corresponding number of events. The string relation at the bottom right is the corresponding string relation. □

Definition 2.3. A $\mathbb{K}$-*annotator* (or *annotator* for short) is a function S that is associated to a finite set $V \subseteq \mathsf{Vars}$ of variables and maps each document d into a $\mathbb{K}$-weighted span relation over V. We denote V by $\mathsf{Vars}(S)$. We sometimes also refer to a $\mathbb{K}$-annotator as an *annotator over* $\mathbb{K}$ when we want to emphasize the semiring.

Example 2.4. As an example of a $\mathbb{K}$-weighted annotator, consider again the setting in Example 2.2. A $\mathbb{Q}$-weighted annotator in this setting is the function S that maps each document d to the span relation R in which the tuples are pairs, consisting of a name of a country and a number (or numeric range), and in which the weight associated to each tuple is the smallest value in the numeric range. An example of such a tuple for the document in Figure 1 would be t_1 with $\mathrm{t}_1(x_{\mathsf{loc}}) = [23, 30\rangle$ (the span of "Belgium") and $\mathrm{t}_1(x_{\mathsf{events}}) = [11, 12\rangle$ (the span of "7"). Another example would be t_2 with $\mathrm{t}_2(x_{\mathsf{loc}}) = [41, 47\rangle$ (the span of "France") and $\mathrm{t}_1(x_{\mathsf{events}}) = [32, 37\rangle$ (the span of "10-15"). The relation R would assign $R(\mathrm{t}_1) = 7$ and $R(\mathrm{t}_2) = 10$. □

We say that two $\mathbb{K}$-annotators S_1 and S_2 are *disjoint* if, for every document $d \in \Sigma^*$, the $\mathbb{K}$-relations $S_1(d)$ and $S_2(d)$ are disjoint. Furthermore, we denote by $S = S'$ the fact that S and S' define the same function.

Notice that $\mathbb{B}$-annotators, i.e., annotators over the Boolean semiring are simply the *functional document spanners* as defined by Fagin et al. [FKRV15a, FKRV15b]. Throughout this article, we refer to $\mathbb{B}$-annotators as *document spanners* (also *spanner* for short).

2.3. **Algebraic Operators on $\mathbb{K}$-Relations and $\mathbb{K}$-Annotators.** Green et al. [GKT07] defined a set of operators on $\mathbb{K}$-relations that naturally correspond to relational algebra operators and map $\mathbb{K}$-relations to $\mathbb{K}$-relations. As in much of the work on semirings in provenance, they do not consider the *difference* operator (which would require additive inverses). More precisely, they define the algebraic operators *union*, *projection*, and *natural join* for all finite sets $V_1, V_2 \subseteq \mathsf{Vars}$ and for all $\mathbb{K}$-relations R_1 over V_1 and R_2 over V_2, as follows.

- **Union**: If $V_1 = V_2$ then the union $R := R_1 \cup R_2$ is a function $R : V_1\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by $R(\mathrm{t}) := R_1(\mathrm{t}) \oplus R_2(\mathrm{t})$. (Otherwise, the union is not defined.)
- **Projection**: For $X \subseteq V_1$, the projection $R := \pi_X(R_1)$ is a function $R : X\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathrm{t}) := \bigoplus_{\mathrm{t}=\pi_X(\mathrm{t}') \text{ and } R_1(\mathrm{t}')\neq\bar{0}} R_1(\mathrm{t}') \,.$$
- **Natural Join:** The natural join $R := R_1 \bowtie R_2$ is a function $R : (V_1 \cup V_2)\text{-}\mathsf{Tup} \to \mathbb{K}$ defined by
$$R(\mathrm{t}) := R_1(\pi_{V_1}(\mathrm{t})) \otimes R_2(\pi_{V_2}(\mathrm{t})) \,.$$

Proposition 2.5 (Green et al. [GKT07])**.** *The above operators preserve the finiteness of the supports. Therefore, they map $\mathbb{K}$-relations into $\mathbb{K}$-relations.*

Hence, we obtain an algebra on $\mathbb{K}$-relations.

We now lift the relational algebra operators on $\mathbb{K}$-relations to the level of $\mathbb{K}$-annotators. For all documents d and for all annotators S_1 and S_2 associated with V_1 and V_2, respectively, we define the following:

- **Union:** If $V_1 = V_2$ then the union $S := S_1 \cup S_2$ is defined by $S(d) := S_1(d) \cup S_2(d)$.[1]
- **Projection:** For $X \subseteq V_1$, the projection $S := \pi_X S_1$ is defined by $S(d) := \pi_X S_1(d)$.
- **Natural Join:** The natural join $S := S_1 \bowtie S_2$ is defined by $S(d) := S_1(d) \bowtie S_2(d)$.

Due to Proposition 2.5, it follows that the above operators form an algebra on $\mathbb{K}$-annotators.

2.4. **Ref-Words.** We use *weighted VSet-automata* (or simply *VSet-automata* for the Boolean semiring) in order to represent $\mathbb{K}$-annotators. Following Freydenberger [Fre19], we introduce so-called *ref-words*, which connect spanner representations with regular languages. We also introduce unambiguous and functional VSet-automata, which have properties essential to the tractability of some problems we study.

For a finite set $V \subseteq \mathsf{Vars}$ of variables, ref-words are defined over the extended alphabet $\Sigma \cup \Gamma_V$, where $\Gamma_V := \{\triangleright_x \mid x \in V\} \cup \{\triangleleft_x \mid x \in V\}$. We assume that Γ_V is disjoint with Σ and Vars. Ref-words extend strings over Σ by encoding opening ($\triangleright_x$) and closing ($\triangleleft_x$) of variables.

A ref-word $\mathrm{r} \in (\Sigma \cup \Gamma_V)^*$ is *valid* if every occurring variable is opened and closed exactly once. More formally, for each $x \in V$, the string r has precisely one occurrence of $\triangleright_x$ and precisely one occurrence of $\triangleleft_x$, which is after the occurrence of $\triangleright_x$. For every valid ref-word r over $(\Sigma \cup \Gamma_V)$, we define $\mathsf{Vars}(\mathrm{r})$ as the set of variables $x \in V$ which occur in the ref-word. More formally,

$$\mathsf{Vars}(\mathrm{r}) \;:=\; \{x \in V \mid \exists \mathrm{r}_x^{\mathrm{pre}}, \mathrm{r}_x, \mathrm{r}_x^{\mathrm{post}} \in (\Sigma \cup \Gamma_V)^* \text{ such that } \mathrm{r} = \mathrm{r}_x^{\mathrm{pre}} \cdot \triangleright_x \cdot \mathrm{r}_x \cdot \triangleleft_x \cdot \mathrm{r}_x^{\mathrm{post}}\}.$$

Intuitively, each valid ref-word r encodes a d-tuple for some document d, where the document is given by symbols from σ in r and the variable markers encode where the spans begin and end. Formally, we define functions doc and tup that, given a valid ref-word, output the corresponding document and tuple.[2] The morphism $\mathrm{doc}\colon (\Sigma \cup \Gamma_V)^* \to \Sigma^*$ is defined on single symbols as:

$$\mathrm{doc}(\sigma) \;:=\; \begin{cases} \sigma & \text{if } \sigma \in \Sigma \\ \varepsilon & \text{if } \sigma \in \Gamma_V \end{cases}$$

and we define $\mathrm{doc}(\sigma_1 \cdots \sigma_n) := \mathrm{doc}(\sigma_1) \cdots \mathrm{doc}(\sigma_n)$.

We now define the function tup. By definition, every valid ref-word r over $(\Sigma \cup \Gamma_V)$ has a unique factorization

$$\mathrm{r} \;=\; \mathrm{r}_x^{\mathrm{pre}} \cdot \triangleright_x \cdot \mathrm{r}_x \cdot \triangleleft_x \cdot \mathrm{r}_x^{\mathrm{post}}$$

for each $x \in \mathsf{Vars}(\mathrm{r})$. We then define the function tup as

$$\mathrm{tup}(\mathrm{r}) \;:=\; \{x \mapsto [i_x, j_x\rangle \mid x \in \mathsf{Vars}(\mathrm{r}), i_x = |\,\mathrm{doc}(\mathrm{r}_x^{\mathrm{pre}})|, j_x = i_x + |\,\mathrm{doc}(\mathrm{r}_x)|\}\;.$$

The usage of the doc morphism in the above definition ensures that the indices i_x and j_x refer to positions in the document and do not consider other variable operations.

A *ref-word language* $\mathcal{R}$ is a language of ref-words. We say that $\mathcal{R}$ is *functional* if every ref-word $\mathrm{r} \in \mathcal{R}$ is valid and there is a set V of variables such that $\mathsf{Vars}(\mathrm{r}) = V$ for each $\mathrm{r} \in \mathcal{R}$.

[1] Here, $\cup$ stands for the union of two K-relations as was defined previously. The same is valid also for the other operators.

[2] The function doc is sometimes also called clr in literature (cf. Freydenberger et al. [FKP18]).

Given a functional ref-word language $\mathcal{R}$, the spanner $[\![\mathcal{R}]\!]$ represented by $\mathcal{R}$ is given by

$$[\![\mathcal{R}]\!](d) \;:=\; \left\{ \mathrm{tup}(\mathrm{r}) \mid \mathrm{r} \in \mathcal{R} \text{ and } \mathrm{doc}(\mathrm{r}) = d \right\} .$$

The Variable Order Condition and the ref *Function.* Let $\mathrm{r} = \triangleright_{x_1} \triangleright_{x_2} a \triangleleft_{x_1} \triangleleft_{x_2}$ and $\mathrm{r}' = \triangleright_{x_1} \triangleright_{x_2} a \triangleleft_{x_2} \triangleleft_{x_1}$ be ref-words. We observe that both ref-words encode the tuple that selects the span $[1, 2\rangle$ in both variables x_1, x_2 on document a. Thus, the same spanner can be represented by multiple ref-word languages. We now introduce the *variable order condition*, in order to achieve a one-to-one mapping between ref-words (resp., ref-word languages) and tuples (resp., spanners). To this end, we fix a total linear order $\prec$ on the set Γ_{Vars} of variable operations, such that $\triangleright_v \prec \triangleleft_v$ for every variable $v \in \mathsf{Vars}$. We say that a ref-word r *satisfies the variable order condition* if all adjacent variable operations in r are ordered according to the fixed linear order $\prec$. That is, the ref-word $\mathrm{r} = \sigma_1 \cdots \sigma_n$ satisfies the variable order condition if $\sigma_i \prec \sigma_{i+1}$ for every $1 \leq i < n$ with $\sigma_i, \sigma_{i+1} \in \Gamma_{\mathsf{Vars}}$. We observe that, for every document d and every tuple and $\mathrm{t} = \mathrm{tup}(\mathrm{r})$ that satisfies the variable order condition.

We define ref as the function that, given a document d and a d-tuple t, returns the unique ref-word that satisfies the variable order condition. The following observation shows the connections between the functions doc, ref, and tup.

Observation 2.6. Let r be a valid ref-word and let $\mathrm{r}' := \mathrm{ref}(\mathrm{doc}(\mathrm{r}), \mathrm{tup}(\mathrm{r}))$. Then $\mathrm{tup}(\mathrm{r}) = \mathrm{tup}(\mathrm{r}')$. Furthermore, $\mathrm{r} = \mathrm{r}'$ if and only if r satisfies the variable order condition.

Analogously to functionality, we say that a ref-word language $\mathcal{R}$ satisfies the variable order condition if every ref-word $\mathrm{r} \in \mathcal{R}$ satisfies the variable order condition.

2.5. **(Weighted) Variable Set-Automata.** In this section, we revisit the definition of *weighted VSet-automaton* as a formalism to represent $\mathbb{K}$-annotators [DKMP22]. This formalism is a natural generalization of both VSet-automata and weighted automata [DKV09]. Throughout the article, we will use weighted VSet-automata for two purposes: we use the VSet-automata over the Boolean semiring $\mathbb{B}$ for extracting spans from documents (as in the usual document spanner framework [FKRV13]) and the more general $\mathbb{K}$-weighted VSet-automata as one formalism for weight functions. (We discuss all considered variants for weight functions in Section 3.3.)

Let $V \subseteq \mathsf{Vars}$ be a finite set of variables. A *weighted variable-set automaton over semiring* $\mathbb{K}$ (alternatively, a *weighted VSet-automaton* or a $\mathbb{K}$*-weighted VSet-automaton*) is a tuple $A := (\Sigma, V, Q, I, F, \delta)$ where Σ is a finite alphabet; $V \subseteq \mathsf{Vars}$ is a finite set of variables; Q is a finite set of *states*; $I : Q \to \mathbb{K}$ is the *initial weight function*; $F : Q \to \mathbb{K}$ is the *final weight function*; and $\delta : Q \times (\Sigma \cup \{\varepsilon\} \cup \Gamma_V) \times Q \to \mathbb{K}$ is a *(*$\mathbb{K}$*-weighted) transition* function. We define the *transitions* of A as the set of triples (p, o, q) with $\delta(p, o, q) \neq \overline{0}$. Likewise, the *initial* (resp., *accepting*) states are those states q with $I(q) \neq \overline{0}$ (resp., $F(q) \neq \overline{0}$). For every semiring element $a \in \mathbb{K}$, we denote the length of the encoding of a by $\|a\|$. The *size* of a weighted VSet-automaton A is defined by

$$|A| := |Q| + \sum_{q \in Q} \|I(q)\| + \sum_{q \in Q} \|F(q)\| + \sum_{p,q \in Q,\ a \in (\Sigma \cup \{\varepsilon\} \cup \Gamma_V)} \|\delta(p, a, q)\| .$$

Runs of A are defined over ref-words. More precisely, a *run* ρ of A on ref-word $\mathrm{r} = \sigma_1 \ldots \sigma_m$ is a sequence $q_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{m-1}} q_{m-1} \xrightarrow{\sigma_m} q_m$ where:

- $I(q_0) \neq \overline{0}$ and $F(q_m) \neq \overline{0}$;

- $\delta(q_i, \sigma_{i+1}, q_{i+1}) \neq \overline{0}$ for all $0 \leq i < m$.

We say that a run ρ is *on a document* d if ρ is a run on r and $\mathrm{doc}(\mathrm{r}) = d$. Furthermore, overloading notation, given a run ρ of A on r, we denote r by $\mathrm{ref}(\rho)$. We define the *ref-word language* $\mathcal{R}(A)$ as the set of all ref-words r such that A has a run on r.

The *weight* of a run is obtained by $\otimes$-multiplying the weights of its constituent transitions. Formally, the weight w_ρ of ρ is an element in $\mathbb{K}$ given by the expression

$$I(q_0) \otimes \delta(q_0, \sigma_1, q_1) \otimes \cdots \otimes \delta(q_m, \sigma_m, q_{m+1}) \otimes F(q_{m+1}) \ .$$

We call ρ *nonzero* if $\mathsf{w}_\rho \neq \overline{0}$. Furthermore, ρ is called *valid* if $\mathrm{ref}(\rho)$ is valid and

$$\mathsf{Vars}(\mathrm{tup}(\mathrm{ref}(\rho))) = V \ .^{3}$$

If ρ is valid we denote the tuple $\mathrm{tup}(\mathrm{ref}(\rho))$ by $\mathrm{tup}(\rho)$.

We say that a weighted VSet-automaton A is *functional* if every run of A is valid. We denote the set of all valid and nonzero runs of A on d by

$$P(A, d) := \{\rho \mid \mathrm{ref}(\rho) \in \mathcal{R}(A) \text{ and } d = \mathrm{doc}(\mathrm{ref}(\rho))\} \ .$$

Notice that there may be infinitely many valid and nonzero runs of a weighted VSet-automaton on a given document, due to *ε-cycles*, which are states $q_1, \ldots, q_k$ such that $(q_i, \varepsilon, q_{i+1})$ is a transition for every $i \in \{1, \ldots, k-1\}$ and $q_1 = q_k$. Following Doleschal et al. [DKMP22] we assume that weighted VSet-automata do not have ε-cycles, unless mentioned otherwise.

As such, if A does not have ε-cycles, then the result of applying A on a document d, denoted $[\![A]\!]_{\mathbb{K}}(d)$, is the $\mathbb{K}$-relation R for which

$$R(\mathrm{t}) := \bigoplus_{\rho \in P(A,d) \text{ and } \mathrm{t} = \mathrm{tup}(\rho)} \mathsf{w}_\rho \ .$$

Note that $P(A, d)$ only contains runs ρ that are valid and nonzero. If t is a V'-tuple with $V' \neq V$ then $R(\mathrm{t}) = \overline{0}$, because we only consider valid runs. In addition, $[\![A]\!]_{\mathbb{K}}$ is a well defined $\mathbb{K}$-annotator since every V-tuple in the support of $[\![A]\!]_{\mathbb{K}}(d)$ is a V-tuple over $\mathsf{Spans}(d)$. To simplify notation, we sometimes denote $[\![A]\!]_{\mathbb{K}}(d)(\mathrm{t})$ — the weight assigned to the d-tuple t by A — by $[\![A]\!]_{\mathbb{K}}(d, \mathrm{t})$. We say that two $\mathbb{K}$-weighted VSet-automata A_1 and A_2 are *disjoint* if $\mathcal{R}(A_1) \cap \mathcal{R}(A_2) = \emptyset$. This implies that also the corresponding $\mathbb{K}$-annotators $[\![A_1]\!]_{\mathbb{K}}$ and $[\![A_2]\!]_{\mathbb{K}}$ are disjoint.

We say that a $\mathbb{K}$-annotator (resp., document spanner) S is *regular* if there exists a weighted VSet-automaton (resp., $\mathbb{B}$-weighted VSet-automaton) A such that $S = [\![A]\!]_{\mathbb{K}}$. Note that this is an equality between functions. If $\mathbb{K}$ is clear from the context, we may just write $[\![A]\!]$ instead of $[\![A]\!]_{\mathbb{K}}$.

We say that two weighted VSet-automata A and A' are *equivalent* if they define the same $\mathbb{K}$-annotator, that is, $[\![A]\!]_{\mathbb{K}} = [\![A']\!]_{\mathbb{K}}$, which is the case if $[\![A]\!]_{\mathbb{K}}(d) = [\![A']\!]_{\mathbb{K}}(d)$ for every $d \in \Sigma^*$.

Similar to our terminology on $\mathbb{B}$-annotators, we refer to (functional) $\mathbb{B}$-weighted VSet-automata as *(functional) VSet-automata*. Since VSet-automata can always be translated into equivalent functional VSet-automata [Fre19, Proposition 3.9], we assume in this article that VSet-automata are functional. This is a common assumption for document spanners involving regular languages [FKRV15a, Fre19, PFKK19]. Furthermore, we assume that all

[3]Note that the second condition ensures that all valid runs are over the same set of variables. This is required, as $\mathbb{K}$-annotators map documents to annotated relations.

weighted VSet-automata are functional as well. In the following, we denote by $\text{REG}_{\mathbb{K}}$ the class of all functional $\mathbb{K}$-weighted VSet-automata and by VSA the class of all functional VSet-automata.

Due to the close relationship between regular expressions and $\mathbb{B}$-weighted automata, and since regular expressions are easy to read, we sometimes define $\mathbb{B}$-weighted VSet-automata using regular expressions over $\Sigma \cup \Gamma_V$. Here, we use $\cdot$ to denote concatenation, $\vee$ to denote disjunction, and * to denote Kleene star. As usual, we often omit $\cdot$ and use priority rules (* before $\cdot$ before $\vee$) for improving the readability of expressions.

Unambiguous (weighted) VSet-Automata. We now discuss unambiguity for (weighted) VSet-automata. A (weighted) VSet-automaton A is *unambiguous* if it satisfies the following two conditions.

(C1) $\mathcal{R}(A)$ satisfies the variable order condition;
(C2) for every $\mathrm{r} \in \mathcal{R}(A)$, there is exactly one run of A on r.

We note that for Boolean spanners, i.e. spanners with no variables, the definitions coincide with the classical unambiguity definition of finite state automata. That is, a VSet-automaton with $\mathsf{Vars}(A) = \emptyset$ is unambiguous if it is a unambiguous finite state automaton. Furthermore, we note that every VSet-automaton A can be transformed to an equivalent unambiguous VSet-automaton A'. (e.g. Doleschal et al. [DKM$^+$21, Lemma 4.5]). However, VSet-automata can be exponentially more succinct than equivalent unambiguous VSet-automata.[4]

Example 2.7. The span relation on the bottom right of Figure 1 can be extracted from d by a spanner that matches textual representations of numbers (or ranges) in the variable x_{events}, followed by a city or country name, matched in x_{loc}. Figure 2 shows how two such VSet-automata may look like. Note that some strings, like Luxembourg are the name of a city as well as a country. Thus, the upper automaton is ambiguous, because the tuple with Luxembourg is captured twice (thus, violating (C2)). The lower automaton is unambiguous, because the sub-automaton for Loc only matches such names once. □

In the following, we denote by $\text{UREG}_{\mathbb{K}}$ the class of $\mathbb{K}$-weighted unambiguous functional VSet-automata and by uVSA the class of unambiguous functional VSet-automata.

2.6. **Aggregate Queries.** Aggregation functions, such as min, max, and sum operate on numerical values from database tuples, whereas all the values of d-tuples are spans. Yet, these spans may represent numerical values, from the document d, encoded by the captured words (e.g., "3," "three," "March" and so on). To connect spans to numerical values, we will use *weight functions*

Definition 2.8 (Weight function)**.** Denote by Tup the set of all V-tuples for sets V, i.e., the union of all sets V-Tup. A *weight function* is a function $w : \Sigma^* \times \mathsf{Tup} \to \{\mathbb{Q} \cup \infty\}$. It maps pairs of documents d and d-tuples t to values in $\mathbb{Q}$ or to ∞.

In the definition of weight functions, we allow the range to include ∞, since we will use subsets of $\mathbb{Q}$ and the tropical semiring $\mathbb{T}$, the latter of which contains ∞. We discuss weight functions in more detail in Section 3.3.

[4]Note that, for functional VSet-automata, the exponential factor in the relative succinctness is caused by condition (C2). That is, for every functional VSet-automaton A, there is an equivalent functional VSet-automaton $|A'|$ which satisfies condition (C1) and is of size at most polynomial in $|A|$.

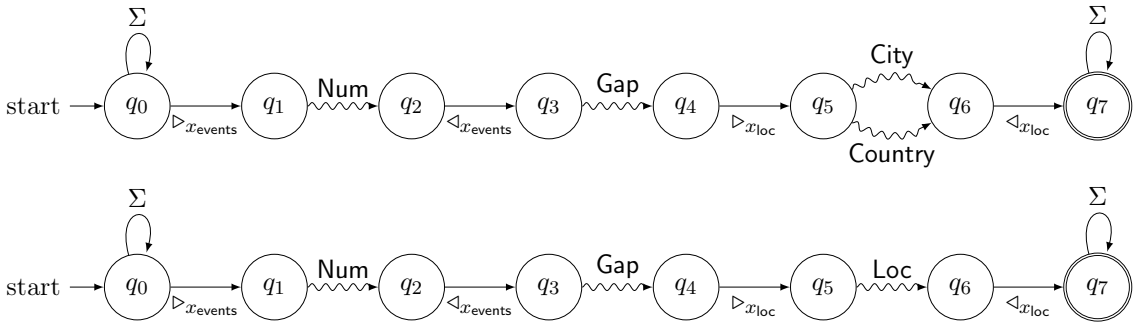

Figure 2: Two example VSet-automata that extract the span relation R on input d as defined in Figure 1. For the sake of presentation, the automata are simplified as follows: Num is a sub-automaton matching anything representing a number (of events) or range, Gap is a sub-automaton matching sequences of at most three words, City and Country are sub-automata matching city and country names respectively. Loc is a sub-automaton for the union of City and Country. All these sub-automata are assumed to be unambiguous.

T	h	e	r	e	␣	a	r	e	␣	7	␣	e	v	e	n	t	s	␣	i	n	␣	B	e	l	g	i	u	m	,	␣	1	0	-	1	5	␣	i	n	␣
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40

F	r	a	n	c	e	,	␣	4	␣	i	n	␣	L	u	x	e	m	b	o	u	r	g	,	␣	t	h	r	e	e	␣	i	n	␣	B	e	r	l	i	n	.
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81

x_{loc}	x_{events}
$[23, 30\rangle$	$[11, 12\rangle$
$[41, 47\rangle$	$[32, 37\rangle$
$[54, 64\rangle$	$[49, 50\rangle$
$[75, 81\rangle$	$[66, 71\rangle$

x_{events}	$W(x_{\text{events}})$
7	7
10-15	10
4	4
three	3

$d_{x_{\text{loc}}}$	$d_{x_{\text{events}}}$	$W_R(\text{t})$
Belgium	7	7
France	10-15	10
Luxembourg	4	4
Berlin	three	3

Figure 3: A document d (top), a span relation R (bottom left), a $\mathbb{Q}$-weighted string relation W (bottom middle) and the $\mathbb{Q}$-weighted string relation W_R resulting from W, d, and R (bottom right).

Example 2.9. Consider the document in Figure 3 and assume that we want to calculate the total number of mentioned events. The relation R at the bottom left depicts a possible extraction of locations with their number of events. The table in the bottom middle depicts a weighted string relation W (where the weight of each string is in the rightmost column). The relation on the bottom right depicts the string relation where each tuple is annotated with a weight corresponding to W, R, and d. To get an understanding of the total number of events, we may want to take the sum over the weights of the extracted tuples, namely $7 + 10 + 4 + 3 = 24$. □

For a spanner S, a document d, and weight function w, we denote by $\text{Img}(S, d, w)$ the set of weights of output tuples of S on d, that is, $\text{Img}(S, d, w) = \{w(d, \text{t}) \mid \text{t} \in S(d)\}$. Furthermore, let $\text{Img}(w) \subseteq \mathbb{Q}$ be the set of weights assigned by w, that is, $k \in \text{Img}(w)$ if and only if there is a document d and a d-tuple t with $w(d, \text{t}) = k$.

Definition 2.10. Let d be a document and A be a VSet-automaton such that $[\![A]\!](d) \neq \emptyset$. Let $S = [\![A]\!]$, let w be a weight function, and $q \in \mathbb{Q}$ with $0 \leq q \leq 1$. We define the following spanner aggregation functions:

$$\begin{aligned}
\mathrm{Count}(S,d) &:= |S(d)| \\
\mathrm{Min}(S,d,w) &:= \min_{\mathrm{t}\in S(d)} w(d,\mathrm{t}) \\
\mathrm{Max}(S,d,w) &:= \max_{\mathrm{t}\in S(d)} w(d,\mathrm{t}) \\
\mathrm{Sum}(S,d,w) &:= \sum_{\mathrm{t}\in S(d)} w(d,\mathrm{t}) \\
\mathrm{Avg}(S,d,w) &:= \frac{\mathrm{Sum}(S,d,w)}{\mathrm{Count}(S,d)} \\
q\text{-Quantile}(S,d,w) &:= \min\left\{ r \in \mathrm{Img}(S,d,w) \;\middle|\; \frac{|\{\mathrm{t} \in S(d) \mid w(d,\mathrm{t}) \leq r\}|}{|S(d)|} \geq q \right\}
\end{aligned}$$

We observe that $\mathrm{Min}(S,d,w) = 0\text{-Quantile}(S,d,w)$ and $\mathrm{Max}(S,d,w) = 1\text{-Quantile}(S,d,w)$.

2.7. **Main Problems.** Let $\mathcal{S}$ be a class of regular document spanners and $\mathcal{W}$ be a class of weight functions. We define the following problems.

Count$[\mathcal{S}]$	
Input:	Spanner $S \in \mathcal{S}$ and document $d \in \Sigma^*$.
Task:	Compute $\mathrm{Count}(S,d)$.

Sum$[\mathcal{S},\mathcal{W}]$	
Input:	Spanner $S \in \mathcal{S}$, document $d \in \Sigma^*$, a weight function $w \in \mathcal{W}$.
Task:	Compute $\mathrm{Sum}(S,d,w)$.

The problems Average$[\mathcal{S},\mathcal{W}]$, q-Quantile$[\mathcal{S},\mathcal{W}]$, Min$[\mathcal{S},\mathcal{W}]$, and Max$[\mathcal{S},\mathcal{W}]$ are defined analogously to Sum$[\mathcal{S},\mathcal{W}]$. Notice that all these problems study *combined complexity*. Since the number of tuples in $S(d)$ is always in $O(|d|^{2k})$, where k is the number of variables of the spanner S (cf. Corlollary 4.6), the *data complexity* of all the problems is in FP: One can just materialize $S(d)$ and apply the necessary aggregate. Under combined complexity, we will therefore need to find ways to avoid materializing $S(d)$ to achieve tractability.

2.8. **Algorithms and Complexity Classes.** Before we discuss our main results in Section 3, we provide a few definitions on computational complexity.

We first define fully polynomial-time randomized approximation schemes (FPRAS).

Definition 2.11. Let f be a function that maps inputs x to rational numbers and let $\mathcal{A}$ be a probabilistic algorithm, which takes an input instance x and a parameter $\delta > 0$. Then $\mathcal{A}$ is called a *fully polynomial-time randomized approximation scheme* (FPRAS), if

- $\Pr\Big(\big|\mathcal{A}(x,\delta) - f(x)\big| \;\leq\; \delta \cdot \big|f(x)\big|\Big) \;\geq\; \frac{3}{4}$;
- the runtime of $\mathcal{A}$ is polynomial in $|x|$ and $\frac{1}{\delta}$.

The following definitions closely follow the Handbook of Theoretical Computer Science [vL91]. The class FP (respectively, FEXPTIME) is the set of all functions that are computable in polynomial time (resp., in exponential time). A *counting Turing Machine* is an non-deterministic Turing Machine whose output for a given input is the number of accepting computations for that input. Given functions $f, g : \Sigma^* \to \mathbb{N}$, f is said to be *parsimoniously reducible* to g in polynomial time if there is a function $h : \Sigma^* \to \Sigma^*$, which is computable in polynomial time, such that for every $x \in \Sigma^*$ it holds that $f(x) = g(h(x))$. Furthermore, we say that f is *Turing reducible* to g in polynomial time, if f can be computed by a polynomial time Turing Machine M, which has access to an oracle for g.

The class #P is the set of all functions that are computable by polynomial-time counting Turing Machines. A problem X is #P-*hard* under parsimonious reductions (resp., Turing reductions) if there are polynomial time parsimonious reductions (resp., Turing reductions) to it from all problems in #P. If in addition $X \in$ #P, we say that X is #P-complete under parsimonious reductions (resp., Turing reductions).

The class $\mathrm{FP}^{\#\mathrm{P}}$ is the set of all functions that are computable in polynomial time by an oracle Turing Machine with a #P oracle. It is easy to see that, under Turing reductions, a problem is hard for the class #P if and only if it is hard for $\mathrm{FP}^{\#\mathrm{P}}$. We note that every problem which is #P-hard under parsimonious reductions is also #P-hard under Turing reductions. Therefore, unless mentioned otherwise, we always use parsimonious reductions.

The class spanL is the class of all functions $f : \Sigma^* \to \mathbb{N}$ for which there is a nondeterministic logarithmic space Turing Machine M with input alphabet Σ such that $f(x) = |M(x)|$.

The class OptP is the set of all functions computable by taking the maximum output value over all accepting computations of a polynomial-time non-deterministic Turing Machine that outputs natural numbers. Assume that Γ is the Turing Machine alphabet. Let $f, g : \Gamma^* \to \mathbb{N}$ be functions. A *metric reduction*, as introduced by Krentel [Kre88], from f to g is a pair of polynomial-time computable functions T_1, T_2, where $T_1 : \Gamma^* \to \Gamma^*$ and $T_2 : \Gamma^* \times \mathbb{N} \to \mathbb{N}$, such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Gamma^*$.

The class BPP is the set of all decision problems solvable in polynomial time by a probabilistic Turing Machine in which the answer always has probability at least $\frac{1}{2} + \delta$ of being correct for some fixed $\delta > 0$.

3. Main Results

In this section we present the main results of this article.

3.1. Known Results.

We begin by giving an overview of the results on Count, which are known from the literature.

Theorem 3.1 Arenas et al. [ACJR19], Florenzano et al. [FRU+18]**.** *Count*[uVSA] *is in* FP *and Count*[VSA] *is* spanL*-complete. Furthermore, Count*[VSA] *can be approximated by an* FPRAS.

Proof. Follows from Arenas et al. [ACJR19, Corollaries 4.1 and 4.2], and Florenzano et al. [FRU+18, Theorem 5.2]. □

Aggregate	Spanner	Weights	Complexity	Approximation
Count	uVSA	-	in FP	-
	VSA	-	#P-hard†	FPRAS
Min	uVSA, VSA	CWidth, UReg, $\text{Reg}_{\mathbb{T}}$	in FP (5.1,7.2)	-
		$\text{Reg}_{\mathbb{Q}}$, Poly	OptP-hard (6.1,7.3)	no FPRAS (8.3)
Max	uVSA, VSA	CWidth, UReg	in FP (5.1,7.2)	-
		$\text{Reg}_{\mathbb{T}}$, $\text{Reg}_{\mathbb{Q}}$, Poly	OptP-hard (6.1,7.3)	no FPRAS (8.3,8.4)
Sum	uVSA	CWidth, UReg, $\text{Reg}_{\mathbb{Q}}$	in FP (5.3,7.4,7.5)	-
		$\text{Reg}_{\mathbb{T}}$, Poly	#P-hard (6.1,7.7)	no FPRAS (8.1)
	VSA	$\text{CWidth}_{\mathbb{N}}$	spanL-complete (5.5)	FPRAS (8.2)
		CWidth, UReg, Reg, Poly	#P-hard (5.4)	no FPRAS (8.1)
Average	uVSA	CWidth, UReg, $\text{Reg}_{\mathbb{Q}}$	in FP (5.3,7.6)	-
		$\text{Reg}_{\mathbb{T}}$, Poly	#P-hard (6.1)	no FPRAS (8.1)
	VSA	$\text{CWidth}_{\mathbb{Q}^+}$	#P-hard† (5.6)	FPRAS (8.8)
		CWidth, UReg, Reg, Poly	#P-hard† (5.6,7.7)	no FPRAS (8.5)
q-Quantile	uVSA	CWidth	in FP (5.3)	-
		UReg,Reg, Poly	#P-hard† (6.2,7.9)	no FPRAS (8.7)
	VSA	CWidth, UReg, Reg, Poly	#P-hard† (5.6)	no FPRAS (8.6)
q-Quantile (positional)	VSA	Poly	-	FPRAS-like approx. (8.10)

Table 1: Detailed overview of complexities of aggregate problems for document spanners. All problems are in FEXPTIME. The "no FPRAS" claims either assume that RP $\neq$ NP or assume that the polynomial hierarchy does not collapse. The #P-hardness results, marked with † rely on Turing reductions. The numbers refer to the numbers of new results.

The spanL lower bound by Florenzano et al. [FRU+18, Theorem 5.2] is due to a parsimonious reduction from the #NFA(n)-problem[5] which is known to be #P-complete under Turing reductions (cf. Kannan et al. [KSM95]). As every parsimonious reduction is also a Turing reduction, the following corollary follows immediately.

Corollary 3.2. *Count*[VSA] *is* #P*-hard under Turing reductions.*

Two observations can be made from these results. First, Count requires the input spanner to be *unambiguous* for tractability. This tractability implies that Count can be computed without materializing the possibly exponentially large set $S(d)$ if the spanner is unambiguous. Furthermore, if the spanner is not unambiguous then, due to spanL-completeness of Count, we do not know an efficient algorithm for its exact computation (and therefore may have to materialize $S(d)$), but Count can be *approximated* by an FPRAS. We will explore to which extent this picture generalizes to other aggregates.

[5]Given an NFA A and a natural number n, encoded in binary, the #NFA(n) problem asks for the number of words $w \in \mathcal{L}(A)$ of length n. The #NFA(n) problem is sometimes also called Census Problem.

3.2. **Overview of New Results.** The complexity results are summarized in Table 1. By now the reader is familiar with the aggregate problems and the types of spanners we study. We obtain different results for different representations of weight functions, which we denote here as CWidth, Poly, and Reg (resp., UReg) and define formally in Section 3.3. Intuitively, CWidth are *constant-width* weight functions that assign values based on strings selected by a constant number of variables; Poly are polynomial-time computable weight functions, and Reg (resp., UReg) are weight functions represented by weighted (resp., unambiguous weighted) VSet-automata. Furthermore, we sometimes restrict these classes based on their range. For instance, $\text{CWidth}_{\mathbb{N}}$ and $\text{CWidth}_{\mathbb{Q}^+}$ are the constant-width weight functions that map to natural numbers and positive rational numbers, respectively.

Entries in the table should be read from left to right. For instance, the third row states that the Min problem, for both spanner classes uVSA and VSA, and for all three classes CWidth, $\text{UReg}_{\mathbb{T}}$, and $\text{Reg}_{\mathbb{T}}$ of weight functions is in FP. Likewise, the fourth row states that the same problems with $\text{Reg}_{\mathbb{Q}}$ or Poly weight functions become OptP-hard and that the existence of an FPRAS would contradict commonly believed conjectures.

In general, the table gives a detailed overview of the impact of (1) unambiguity of spanners and (2) different weight function representations on the complexity of computing aggregates.

3.3. **Results for Different Weight Functions.** We formalize how we represent the weight functions for our new results. Recall that weight functions w map pairs consisting of a document d and d-tuple t to values in $\mathbb{Q} \cup \{\infty\}$.

3.3.1. *Constant-Width Weight Functions.* The simplest type of weight functions we consider are the *constant-width weight functions*.[6] Let $1 \leq c \in \mathbb{N}$ be a constant. A *constant-width weight function (CWidth)* w assigns values based on the strings selected by at most c variables. A constant-width weight function CWidth is given in the input as a $\mathbb{Q}$-weighted string relation, i.e., a string relation R over the numerical semiring $\mathbb{Q} = (\mathbb{Q}, +, \times, 0, 1)$ and the variables X, where $X \subseteq \mathsf{Vars}$, is a set of at most c variables. Recall that d_{t} denotes the tuple $(d_{\mathrm{t}(x_1)}, \ldots, d_{\mathrm{t}(x_n)})$, where $\mathsf{Vars}(\mathrm{t}) = \{x_1, \ldots, x_n\}$. To facilitate presentation, we assume that the variables in X are always present in t, that is, $X \subseteq \mathsf{Vars}(\mathrm{t})$. The weight function $w(d, \mathrm{t})$ is defined as

$$w(d, \mathrm{t}) = R(d_{\pi_X \mathrm{t}}) \ .$$

As we will see in Section 5, the problems Max[VSA, CWidth] and Min[VSA, CWidth] are in FP (Theorem 5.1). Furthermore, we show that the problems Sum[$\mathcal{S}$, CWidth], Average[$\mathcal{S}$, CWidth], and q-Quantile[$\mathcal{S}$, CWidth] behave similarly to Count[$\mathcal{S}$], that is, they are in FP if $\mathcal{S} =$ uVSA (Theorem 5.3) and intractable if $\mathcal{S} =$ VSA (Theorems 5.4, 5.5, and 5.6).

[6] These generalize the single-variable weight functions of Doleschal et al. [DBKM21].

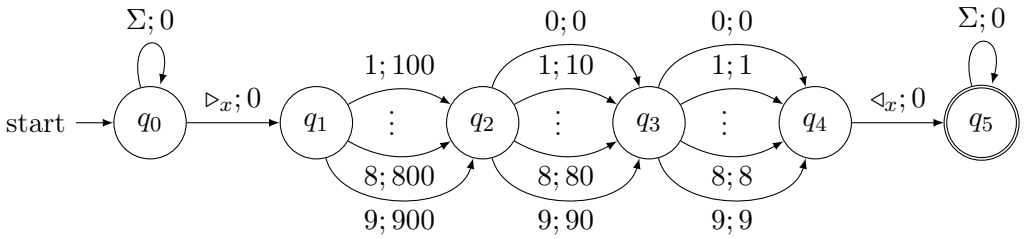

Figure 4: An unambiguous weighted VSet-automaton over the tropical semiring with initial state q_0 (with weight 0) and accepting state q_5 (with weight 0), extracting three-digit natural numbers captured in variable x. Recall that, over the tropical semiring, the weight of a run is the sum of all its edge weights.

3.3.2. *Polynomial-Time Weight Functions.* How far can we push our tractability results? Next, we consider more general ways of mapping d-tuples into numbers. The most general class of weight functions we consider is the set of *polynomial-time weight functions* (Poly). A function w from Poly is given in the input as a polynomial-time Turing Machine M that maps (d, t)-pairs to values in $\mathbb{Q}$ and defines $w(d, \mathrm{t}) = M(d, \mathrm{t})$. Not surprisingly there are multiple drawbacks of having arbitrary polynomial time weight functions. The first is that all considered aggregates become intractable, even if we only consider unambiguous VSet-automata (Theorems 6.1, and 6.2). However, all aggregates can at least be computed in exponential time (Theorem 6.3).

3.3.3. *Regular Weight Functions.* As the class of polynomial-time weight functions quickly leads to intractability, we focus on a restricted class Reg that we introduce next and is less restrictive than CWidth but not as general as Poly such that we can understand the structure of the representation towards efficient algorithms.[7] Our final classes of weight functions are based on $\mathbb{K}$-Annotators. More precisely, we consider weighted VSet-automata and unambiguous weighted VSet-automata over the tropical semiring $\mathbb{T} = (\mathbb{Q} \cup \{\infty\}, \min, +, \infty, 0)$ and the numerical semiring $\mathbb{Q} = (\mathbb{Q}, +, \times, 0, 1)$.[8] Formally, let $\mathrm{Reg} := \mathrm{Reg}_{\mathbb{T}} \cup \mathrm{Reg}_{\mathbb{Q}}$ be the class of all annotators over the tropical or numerical semiring. A *regular* (Reg) weight function w is represented by a weighted VSet-automaton W and defines $w(d, \mathrm{t}) = [\![W]\!](d, \pi_{\mathsf{Vars}(W)}(\mathrm{t}))$. Furthermore, as for constant width weight functions, we assume that the variables used by W are always present in t, that is, $\mathsf{Vars}(W) \subseteq \mathsf{Vars}(\mathrm{t})$.

The set of *unambiguous regular* (UReg) weight functions is the subset of Reg that is represented by unambiguous weighted VSet-automata, that is $\mathrm{UReg} := \mathrm{UReg}_{\mathbb{T}} \cup \mathrm{UReg}_{\mathbb{Q}}$.

Example 3.3. Figure 4 gives an unambiguous weighted VSet-automaton over the tropical semiring that extracts the values of three-digit natural numbers from text. It can easily be extended to extract natural numbers of up to a constant number of digits by adding nondeterminism. Likewise, it is possible to extend it to extract weights as in Example 2.9. If a single variable captures a list of numbers, similar to $d_{[32,37\rangle} = 10{-}15$, one may use ambiguity to extract the minimal number represented in this range. □

[7] We prove in Section 4.2 that CWidth ⊆ Reg ⊆ Poly; also see Figure 5.

[8] One can also consider the tropical semiring with max/plus, in which case the complexity results are analogous to the ones we have for the tropical semiring with min/plus, with Min and Max interchanged.

Our results for regular and unambiguous regular weight functions are similar to CWidth when it comes to Min, Max, Sum, and Average. The main difference is that, depending on the semiring, we require more unambiguity. For instance, for the tropical semiring, one needs unambiguity of the regular weight function for Max. For Sum and Average one needs unambiguity for *both* the spanner and the regular weight function to achieve tractability. Contrary, over the numerical semiring, one needs unambiguity of the regular weight function for Min and Max, whereas for Sum and Average unambiguity of the spanner is sufficient for tractability. For q-Quantile, the situation is different from CWidth in the sense that regular weight functions render the problem intractable. We refer to Table 1 for an overview.

3.4. **Approximation.** In the cases where exact computation of the aggregate problem is intractable, we consider the question of approximation. It turns out that there exist FPRAS's in two settings that we believe to be interesting. Firstly, in the case of Sum and Average and constant-width weight functions, the restriction of unambiguity in the spanner can be dropped if the weight function uses only nonnegative weights. Secondly, although q-Quantile is #P-hard under Turing reductions for general VSA, it is possible to *positionally* approximate the Quantile element in an FPRAS-like fashion, even with the very general polynomial-time weight functions. We discuss this problem in more detail in Section 8.

4. Preliminary Results

In this section, we give basic results for document spanners and weight functions that we use throughout this article.

4.1. **Known Results on $\mathbb{K}$-Annotators.** We begin by recalling some known results on $\mathbb{K}$-annotators.

Proposition 4.1 (Doleschal et al. [DKMP22, Proposition 6.1])**.** *For every weighted VSet-automaton A there is an equivalent weighted VSet-automaton A' that has no ε-transitions. This automaton A' can be constructed from A in polynomial time. Furthermore, A is functional if and only if A' is functional.*

The following Theorem follows directly from Doleschal et al. [DKMP22, Theorem 6.4] and Doleschal [Dol21, Theorem 5.5.4, Lemma 5.5.5 and Lemma 5.5.9].

Theorem 4.2. *Let $A_1, A_2 \in \mathit{REG}_{\mathbb{K}}$ be $\mathbb{K}$-weighted functional VSet-automata and $X \subseteq \mathsf{Vars}(A_1)$. Then, $A_\pi, A_\cup, A_\bowtie \in \mathit{REG}_{\mathbb{K}}$ can be constructed in polynomial time, such that*

$$[\![A_\cup]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \cup [\![A_2]\!]_{\mathbb{K}}$$
$$[\![A_\pi]\!]_{\mathbb{K}} = \pi_X [\![A_1]\!]_{\mathbb{K}}$$
$$[\![A_\bowtie]\!]_{\mathbb{K}} = [\![A_1]\!]_{\mathbb{K}} \bowtie [\![A_2]\!]_{\mathbb{K}}.$$

Furthermore, $A_\bowtie \in \mathit{UREG}_{\mathbb{K}}$, if $A_1, A_2 \in \mathit{UREG}_{\mathbb{K}}$, and $A_\cup \in \mathit{UREG}_{\mathbb{K}}$ if $A_1, A_2 \in \mathit{UREG}_{\mathbb{K}}$ and $\mathcal{R}(A_1) \cap \mathcal{R}(A_2) = \emptyset$.

4.2. **Relative Expressiveness of Weight Functions.** We first show that every constant-width weight function is also an unambiguous regular weight function.

Proposition 4.3. *CWidth* $\subseteq$ *UReg*$_{\mathbb{Q}} \cap$ *UReg*$_{\mathbb{T}}$.

Proof. Let $w \in$ CWidth be a constant-width weight function, represented by a $\mathbb{Q}$-weighted string relation R over X, that is, tuples in R map variables to strings. We begin by showing that $w \in \text{UReg}_{\mathbb{Q}}$. Let $X = \{x_1, \ldots, x_n\}$. We construct a $\mathbb{Q}$-annotator W representing w. We define an unambiguous VSet-automaton A_{t}, for every tuple $\mathrm{t} \in R$, such that $\mathrm{t}' \in [\![A_{\mathrm{t}}]\!]_{\mathbb{B}}(d)$ if and only if $d_{\mathrm{t}'} = \mathrm{t}$. Let $\mathrm{t} \in R$. For every $x \in X$, let w_x be the word $\mathrm{t}(x)$ and let

$$A_{\mathrm{t}}^x := \Sigma^* \cdot \triangleright_x w_x \triangleleft_x \cdot \Sigma^* \ ,$$

that is, A_{t}^x matches the string $\mathrm{t}(x)$ in variable x and outputs the corresponding $\{x\}$-tuple with the span. Since our definition of unambiguity requires one run per *ref-word* in the language, it is easy to see that such an unambiguous A_{t}^x exists. Furthermore,

$$A_{\mathrm{t}} := A_{\mathrm{t}}^{x_1} \bowtie \cdots \bowtie A_{\mathrm{t}}^{x_n} \ ,$$

which is unambiguous due to Theorem 4.2.

We define W_{t} as the unambiguous $\mathbb{Q}$-weighted VSet-automaton such that

$$[\![W_{\mathrm{t}}]\!]_{\mathbb{Q}}(d, \mathrm{t}') = \begin{cases} R(\mathrm{t}) & \text{if } d_{\mathrm{t}'} = \mathrm{t} \\ \overline{0} & \text{otherwise.} \end{cases}$$

This can be achieved by interpreting A_{t} as a $\mathbb{Q}$-weighted VSet-automaton, where all edges have weight $\overline{1}$, the final weight function assigns weight $\overline{1}$ to all accepting states, and the initial weight function assigns weight $R(\mathrm{t})$ to the initial state of A_{t}. We finally define W as the union of all W_{t}. That is,

$$W = \bigcup_{\mathrm{t} \in R} W_{\mathrm{t}} \ .$$

We observe that, by Theorem 4.2, W must be unambiguous, as all W_{t} are unambiguous and the ref-word languages of the automata W_{t} are pairwise disjoint.

Recall that $[\![W]\!]_{\mathbb{Q}}(d, \mathrm{t}) = \overline{0} = 0$ if there is no run of W on $\mathrm{ref}(d, \mathrm{t})$, i.e. $d_{\mathrm{t}} \notin R$. Therefore, $[\![W]\!]_{\mathbb{Q}}(d, \mathrm{t}) = R(d_{\mathrm{t}})$ as desired.

The proof for CWidth $\subseteq \text{UReg}_{\mathbb{T}}$ follows the same lines. However, the zero element of the tropical semiring is ∞, which implies that the automaton W must have exactly one run ρ for every tuple t, even if $w(d, \mathrm{t}) = 0$. To this end, let W_{t} be as defined before, but interpreted over the tropical semiring. We construct an unambiguous $\mathbb{T}$-weighted VSet-automaton $W_{\overline{R}}$, such that $[\![W_{\overline{R}}]\!]_{\mathbb{T}}(d, \mathrm{t}) = 0$ if $d_{\mathrm{t}} \notin R$ and $W_{\overline{R}}$ has no run for t otherwise. We observe that R is a recognizable string relation.[9] Therefore, due to Doleschal et al. [DKMP22, Theorem 6.11], there is a document spanner A_R, with $\mathrm{t} \in [\![A_R]\!](d)$ if and only if $d_{\mathrm{t}} \in R$. Furthermore, let $A_{\overline{R}}$ be the complement of A_R, that is, $\mathrm{t} \in [\![A_{\overline{R}}]\!](d)$ if and only if $d_{\mathrm{t}} \notin R$. Note that $A_{\overline{R}} \in$ VSA as regular document spanners are closed under difference (cf. Fagin et al. [FKRV15a, Theorem 5.1]). By Doleschal et al. [DKM$^+$21, Lemma 4.5], we can assume w.l.o.g. that $A_{\overline{R}} \in$ uVSA. Let $W_{\overline{R}}$ be $A_{\overline{R}}$, interpreted as $\mathbb{T}$-weighted VSet-automaton, that is, each transition, initial

[9] A k-ary string relation is recognizable if it is a finite union of Cartesian products $L_1 \times \cdots \times L_k$, where each L_i is a regular language. Note that R is recognizable as it is the union over all tuples $\mathrm{t} \in R$, where each tuple is represented by the Cartesian product $\{\mathrm{t}(x_1)\} \times \cdots \times \{\mathrm{t}(x_n)\}$ with $\mathsf{Vars}(\mathrm{t}) = \{x_1, \ldots, x_n\}$.

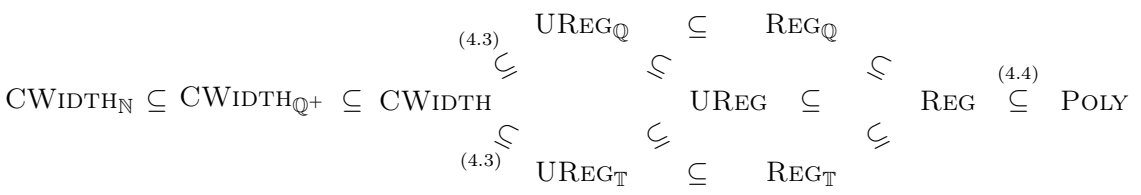

Figure 5: Inclusion structure of our considered weight functions

and final state gets weight $\overline{1} = 0$. Note that, due to $A_{\overline{R}} \in$ uVSA, $W_{\overline{R}}$ is unambiguous. It follows that $[\![W_{\overline{R}}]\!]_{\mathbb{T}}(d, \mathrm{t}) = 0$ if $d_{\mathrm{t}} \notin R$ and $W_{\overline{R}}$ has no run for t otherwise. Let

$$W = W_{\overline{R}} \cup \bigcup_{\mathrm{t} \in R} W_{\mathrm{t}} \,.$$

Again, we observe that, by Theorem 4.2, W must be unambiguous as all involved automata are unambiguous and their ref-word languages are pairwise disjoint. Furthermore,

$$[\![W]\!]_{\mathbb{T}}(d, \mathrm{t}) = \begin{cases} R(d_{\mathrm{t}}) & \text{if } d_{\mathrm{t}} \in R \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $[\![W]\!]_{\mathbb{T}}(d, \mathrm{t}) = R(d_{\mathrm{t}})$ as desired. □

We now observe that every regular weight function is a polynomial-time weight function. Indeed, given a document d and a d-tuple t, the weight $w(d, \mathrm{t})$ for a regular weight function w can be computed in polynomial time (cf. Doleschal [Dol21, Theorem 5.6.1]).

Observation 4.4. REG ⊆ POLY.

To summarize, we provide the inclusion structure of the classes of weight functions we consider in Figure 5. All inclusions that do not have a number hold by definition.

4.3. **Preliminary Results on Document Spanners.** We will also need some preliminary results concerning the number of possible spans over a document d.

Lemma 4.5. *Given a document d, the number of spans over d is polynomial in the size of d. In particular, $|\mathsf{Spans}(d)| = \frac{(|d|+1)\cdot(|d|+2)}{2}$, for every $d \in \Sigma^*$.*

Proof. For a span $[i, j\rangle$, let $\ell = j - i$ be the length of the span. It is easy to see that for every document d, there is exactly one span of length $|d|$, two spans of length $|d| - 1$, three spans of length $|d| - 2$, etc. Thus, there are $1 + 2 + \cdots + (|d| + 1) = \frac{(|d|+1)\cdot(|d|+2)}{2}$ spans over a document d, concluding the proof. □

It follows directly that the maximal number of tuples, extracted by a document spanner is exponential in the size of the spanner.

Corollary 4.6. *Let $A \in$ VSA be a VSet-automaton and $d \in \Sigma^*$ be a document. Then* $\mathrm{Count}(S, d) \leq |\mathsf{Spans}(d)|^{|\mathsf{Vars}(A)|} = \left(\frac{(|d|+1)\cdot(|d|+2)}{2}\right)^{|\mathsf{Vars}(A)|}$.

As we see next, given a number of variables, a document d, and a number k of tuples, we can construct an unambiguous VSet-automaton A and a document d' such that A extracts exactly k tuples on d'.

Lemma 4.7. *Let $X := \{x_1, \ldots, x_v\} \in \mathsf{Vars}$ be a set of variables, $d \in \Sigma^*$ be a document, and $0 \leq k \leq |\mathsf{Spans}(d)|^{|X|}$. Then there is a VSet-automaton $A \in$ uVSA with $\mathsf{Vars}(A) = X$ and a document $d' \in \Sigma^*$ such that $|[\![A]\!](d')| = k$. Furthermore, A and d' can be constructed in time polynomial in $|X|$ and d.*

Proof. We observe that the statement holds for $k = 0$. Therefore we assume, w.l.o.g., that $1 \leq k \leq |\mathsf{Spans}(d)|^v$.

We begin by proving the statement for $|X| = 1$. Let $1 \leq k \leq |\mathsf{Spans}(d)|$. Recalling the proof of Lemma 4.5, we observe that k can be written as a sum $k = k_1 + \cdots + k_n$ of $n \leq |d|+1$ different natural numbers with $0 \leq k_1 < \cdots < k_n \leq |d| + 1$. We construct an automaton $A_k \in$ uVSA, which consists of n branches, corresponding to $k_1, \ldots, k_n$. On document d, the branch corresponding to k_i selects all spans of length $\ell_i := |d| + 1 - k_i$. Each of these branches can be constructed as an unambiguous VSet-automaton $A_{k_i} := \Sigma^* \cdot \triangleright_x \Sigma^{\ell_i} \triangleleft_x \cdot \Sigma^*$. We observe that there are exactly k_i spans over d with length ℓ_i, and therefore $|[\![A_{k_i}]\!](d)| = k_i$. The automaton A_k is defined as

$$A_k := A_{k_1} \cup \cdots \cup A_{k_n} \ .$$

It is straightforward to verify that all automata A_{k_i} are unambiguous. Thus, since the ref-word languages of all A_{k_i} are pairwise disjoint, it holds that $A_k \in$ uVSA (cf. Theorem 4.2). Furthermore, we observe that

$$|[\![A_k]\!](d)| = |[\![A_{k_1}]\!](d)| + \cdots + |[\![A_{k_n}]\!](d)| = k_1 + \cdots + k_n = k \ .$$

It remains to show the statement for $v := |X| > 1$. Let $\# \notin \Sigma$ be a new alphabet symbol. We build upon the encoding for $|X| = 1$. That is, for every $1 \leq k \leq |\mathsf{Spans}(d)|$, let A_k^x be the automaton A_k, using variable x, as defined previously. We observe that every $1 \leq k \leq |\mathsf{Spans}(d)|^v$ has an encoding $k = k_1 \cdots k_v$ in base $|\mathsf{Spans}(d)|$ of length v. The document d' consists of v copies of $d \cdot \#$, more formally,

$$d' := (d \cdot \#)^v \ .$$

For every $1 \leq i \leq v$, we construct an automaton A'_{k_i}, which selects exactly $k_i \cdot |\mathsf{Spans}(d)|^{v-i}$ tuples over document d'. More formally,

$$A'_{k_i} := d \cdot \triangleright_{x_1} \# \triangleleft_{x_1} \cdot d \cdot \triangleright_{x_2} \# \triangleleft_{x_2} \cdots d \cdot \triangleright_{x_{i-1}} \# \triangleleft_{x_{i-1}} \cdot A_{k_i}^{x_i} \cdot \# \cdot A_{|\mathsf{Spans}(d)|}^{x_{i+1}} \cdot \# \cdots \# \cdot A_{|\mathsf{Spans}(d)|}^{x_v} \cdot \# \ .$$

The automaton A'_k is then defined as the union of all A'_{k_i}, that is,

$$A'_k := A'_{k_1} \cup \cdots \cup A'_{k_v} \ .$$

We observe that $A'_{k_i} \in$ uVSA and due to the ref-word languages of all A'_{k_i} being pairwise disjoint, $A'_k \in$ uVSA (cf. Theorem 4.2). Furthermore, we observe that

$$|[\![A'_k]\!](d')| = |[\![A'_{k_1}]\!](d')| + \cdots + |[\![A'_v]\!](d)'| = k_1 + \cdots + k_n = k \ .$$

This concludes the proof. □

5. Constant-Width Weight Functions

We begin this section by showing that Min and Max are tractable for constant-width weight functions. The reason for their tractability is that, for a constant number of variables $X \subseteq \mathsf{Vars}(A)$, the spans associated to X in output tuples can be computed in polynomial time. Building upon Corollary 4.6, we show that Min and Max are in FP for constant-width weight functions and VSet-automata. We immediately have:

Theorem 5.1. *Min*[VSA, *CWidth*] *and Max*[VSA, *CWidth*] *are in* FP.

Proof. Let $A \in \mathrm{VSA}$, $d \in \Sigma^*$, $X \subseteq \mathsf{Vars}(A)$ with $|X| \leq c$, and $w \in \mathrm{CWidth}$ be given as a $\mathbb{Q}$-weighted string relation R over X. We first show that the set $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\}$ can be computed in time polynomial in the sizes of A and d.

We observe that, per definition of projection for document spanners (Section 2.3), $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\} = \big(\pi_X([\![A]\!])\big)(d)$. Since A is functional (which we assume for VSet-automata throughout this article), a VSet-automaton for $\pi_X([\![A]\!])$ can be computed in polynomial time (cf. Freydenberger et al. [FKP18, Lemma 3.8]). Due to $|X| \leq c$, it follows from Corollary 4.6 that there are at most polynomially many tuples in $\big(\pi_X([\![A]\!])\big)(d)$. Thus, the set $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\}$ can be materialized in polynomial time.

In order to compute Min and Max, a polynomial time algorithm can iterate over all tuples t in $\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\}$, evaluate $R(d, \mathrm{t})$ and maintain the minimum and the maximum of these numbers. □

In order to calculate aggregates like Sum, Avg, or q-Quantile, it is not sufficient to know which weights are assigned, but also the multiplicity of each weight is necessary. Recall that counting the number of output tuples is tractable if the VSet-automaton is unambiguous (Theorem 3.1) and spanL-complete in general. We now show that we can achieve tractability of the mentioned aggregate problems if the VSet-automaton is unambiguous. The reason is that we can compute in polynomial time the multiset $\mathcal{S}_{A,d} := \{\!\{\pi_X \mathrm{t} \mid \mathrm{t} \in [\![A]\!](d)\}\!\}$, where we represent the multiplicity of each tuple t′ (i.e., the number of tuples $\mathrm{t} \in [\![A]\!](d)$ such that $\pi_X \mathrm{t} = \mathrm{t}'$) in binary.

Lemma 5.2. *Given a VSet-automaton A and a document d, the multiset $\mathcal{S}_{A,d}$ can be computed in* FP *if* $A \in \mathrm{uVSA}$.

Proof. The procedure is given as Algorithm 1. It is straightforward to verify that the algorithm is correct. Due to Corollary 4.6, the set $\pi_X([\![A]\!])(d)$ is at most of polynomial size. Furthermore, the automaton $A_{\mathrm{ref}(d,\mathrm{t})} := \mathrm{ref}(d, \mathrm{t}) \in \mathrm{uVSA}$ can be constructed in polynomial time and due to Theorem 4.2 an unambiguous VSet-automaton for A_{t} can be computed in polynomial time as well. By Theorem 3.1, each iteration of the for-loop also only requires polynomial time. Thus, the whole algorithm terminates after polynomially many steps. □

It follows that all remaining aggregate functions can be efficiently computed if the spanner is given as an unambiguous VSet-automaton.

Theorem 5.3. *For every $0 \leq q \leq 1$, Sum*[uVSA, *CWidth*]*, Average*[uVSA, *CWidth*]*, and q-Quantile*[uVSA, *CWidth*] *are in* FP.

Proof. Let $A \in \mathrm{uVSA}$ be a VSet-automaton, $d \in \Sigma^*$ be a document, $w \in \mathrm{CWidth}$ be a weight function, represented by a $\mathbb{Q}$-weighted string relation R over X. Due to Lemma 5.2 the multiset $\mathcal{S}_{A,d}$ can be computed in polynomial time. Thus one can compute the multiset

Algorithm 1: Calculate the multiset $\mathcal{S}_{A,d}$.

Input: An unambiguous VSet-automaton $A \in \text{uVSA}$, a document $d \in \Sigma^*$.
Output: The multiset $\mathcal{S}_{A,d}$.

```
S ← {{ }}
S ← π_X([[A]])(d)
for t ∈ S do
    A_t ← A ⋈ A_ref(d,t)        ▷ A_ref(d,t) is the uVSA that only accepts ref(d, t).
    S(π_X t) ← Count([[A_t]], d)
output S
```

$W := \{\!\{R(d_\mathrm{t}) \mid \mathrm{t} \in \mathcal{S}_{A,d}\}\!\}$ in polynomial time. It is straightforward to compute the aggregates in polynomial time from W. □

We conclude this section by showing that Sum, Avg, and q-Quantile are not tractable, if the spanner is given as a VSet-automaton.

Theorem 5.4. *Sum*[VSA, *CWidth*] *is* #P*-hard, even if w is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with*

$$R(d) := \begin{cases} 1 & \text{if } d = 1 \\ -1 & \text{if } d = -1 \\ 0 & \text{otherwise.} \end{cases}$$

Proof. We will give a reduction from the #CNF problem, which is #P-complete under parsimonious reductions. To this end, let ϕ be a Boolean formula in CNF over variables $x_1, \ldots, x_n$ and let $w \in \text{CWidth}$ be the weight function which is represented by the $\mathbb{Q}$-Relation R, which is as defined in the theorem statement.

We construct a VSet-automaton $A \in \text{VSA}$ and a document $d := a^n \cdot - \cdot 1$, such that $\text{Sum}([\![A]\!], d, w) = c$, where c is the number of variable assignments which satisfy ϕ.

We begin by defining two VSet-automata A_1, A_{-1}, with $\mathsf{Vars}(A_1) = \mathsf{Vars}(A_{-1}) = \{x_1, \ldots, x_n, x\}$. Slightly overloading notation, we define both automata by regex formulas.

The automaton A_1 selects exactly 2^n tuples on document d, all of which get assigned weight 1 by w. More formally (using $\vee$ to denote regular expression disjunction),

$$A_1 := (\triangleright_{x_1} a \triangleleft_{x_1} \ \vee \ \triangleright_{x_1} \varepsilon \triangleleft_{x_1} a) \cdots (\triangleright_{x_n} a \triangleleft_{x_n} \ \vee \ \triangleright_{x_n} \varepsilon \triangleleft_{x_n} a) \ - \ \triangleright_x 1 \triangleleft_x \ .$$

Therefore, $\text{Sum}([\![A_1]\!], d, w) = \text{Count}([\![A_1]\!], d) = 2^n$.

We use a similar encoding as Doleschal et al. [DKMP22, Theorem 5.4] to encode variable assignments into tuples. That is, each variable x_i of ϕ is associated with a corresponding capture variable x_i of A_{-1}. With each assignment τ we associate the tuple t_τ, such that

$$\mathrm{t}_\tau(x_i) := \begin{cases} [i, i\rangle & \text{if } \tau(x_i) = 0, \text{ and} \\ [i, i+1\rangle & \text{if } \tau(x_i) = 1 \ . \end{cases}$$

We construct the automaton A_{-1} as a regex formula α, such that there is a one-to-one correspondence between the non-satisfying assignments for ϕ and tuples in $[\![\alpha]\!](d)$. More

formally, for each clause C_j of ϕ and each variable x_i, we construct a regex-formula

$$\alpha_{i,j} := \begin{cases} x_i\{\varepsilon\} \cdot a & \text{if } x_i \text{ appears in } C_j, \\ x_i\{a\} & \text{if } \neg x_i \text{ appears in } C_j, \\ (x_i\{\varepsilon\} \cdot a) \vee x_i\{a\} & \text{otherwise.} \end{cases}$$

Consequently, we define $\alpha_j := \alpha_{1,j} \cdots \alpha_{n,j} \cdot \triangleright_x 1 \triangleleft_x$.

For example, if we use variables x_1, x_2, x_3, x_4 and $C_j = x_1 \vee x_3 \vee \neg x_4$ is a clause, then

$$\alpha_j = \triangleright_{x_1} \varepsilon \triangleleft_{x_1} \;\; a \; (\triangleright_{x_2} \varepsilon \triangleleft_{x_2} \;\; a \vee \triangleright_{x_2} a \triangleleft_{x_2}) \;\; \triangleright_{x_3} \varepsilon \triangleleft_{x_3} \;\; a \;\; \triangleright_{x_4} a \triangleleft_{x_4} \;\; \triangleright_x 1 \triangleleft_x \, .$$

We observe that $\mathrm{t} \in [\![\alpha_j]\!](d)$ if and only if the variable assignment τ of ϕ with $\mathrm{t} = \mathrm{t}_\tau$ does not satisfy clause C_j.

We finally define $\alpha := \alpha_1 \vee \cdots \vee \alpha_m$, that is, the disjunction of all α_i and A_{-1} as the VSet-automaton corresponding to α.[10] Therefore, $\mathrm{Count}([\![A_{-1}]\!], d) = s$, where $s = 2^n - c$ is the number of variable assignments which do not satisfy ϕ. Furthermore, per definition of A_{-1} and w, it follows that

$$\mathrm{Sum}([\![A_{-1}]\!], d, w) = -1 \cdot s = -s \, .$$

We finally define the VSet-automaton A as the union of A_1 and A_{-1}. We observe that every tuple $\mathrm{t} \in [\![A]\!](d)$ is either selected by A_1 (if $d_{\mathrm{t}(x)} = 1$) or by A_{-1} (if $d_{\mathrm{t}(x)} = -1$), but never by both automata. Recall that c is the number of assignments which satisfy ϕ and $s = 2^n - c$ is the number non-satisfying assignments of ϕ. Therefore, we have that

$$\mathrm{Sum}([\![A]\!], d, w) = \mathrm{Sum}(A_1, d, w) + \mathrm{Sum}(A_{-1}, d, w) = 2^n + (-s) = 2^n - (2^n - c) = c \, .$$

This concludes the proof. □

If the weights are restricted to natural numbers, Sum becomes spanL-complete. Note that we restrict weight functions to natural numbers, because spanL is a class of functions that return natural numbers. Allowing positive rational numbers does not fundamentally change the complexity of the problems though. We will see in Section 8 that this enables us to approximate Sum aggregates.

Theorem 5.5. *Sum*[VSA, *CWidth*$_{\mathbb{N}}$] *is* spanL*-complete, even if w is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with*

$$R(d) := \begin{cases} 1 & \textit{if } d = 1 \\ 0 & \textit{otherwise.} \end{cases}$$

Proof. Recall that a function f is in spanL, if there is an NL Turing machine M such that $f(x) = |M(x)|$. Let $A \in \mathrm{VSA}$ be a VSet-automaton, $d \in \Sigma^*$ be a document, and $w \in \mathrm{CWidth}_{\mathbb{Q}_+}$ be a weight function. We define M as the Turing machine, which guesses a d-tuple t and checks whether $\mathrm{t} \in [\![A]\!](d)$. If yes, M computes the weight $w(d, \mathrm{t})$, which can be done in NL, since w is given by a $\mathbb{Q}$-Relation. The Turing machine M then branches into $w(d, \mathrm{t})$ accepting branches. If $\mathrm{t} \notin [\![A]\!](d)$, M rejects. Thus, $|M(A, d)| = \mathrm{Sum}(S, d, w)$, and therefore Sum[VSA, $\mathrm{CWidth}_{\mathbb{N}}$] is in spanL.

For the lower bound, we give a reduction from Count[VSA], which is spanL-complete (cf. Theorem 3.1). Let $A \in \mathrm{VSA}$, $d \in \Sigma^*$. We assume, w.l.o.g., that $1 \notin \Sigma$ and $x \notin \mathsf{Vars}(A)$. We construct a document $d' := d \cdot 1$ and a VSet-automaton $A' := A \cdot \triangleright_x 1 \triangleleft_x$. We observe that $\mathrm{Sum}([\![A']\!], d', w) = \mathrm{Count}([\![A]\!], d)$, concluding the proof. □

[10] It is easy to verify that the automaton $A_{-1} \in \mathrm{VSA}$ can be constructed in polynomial time from α.

We conclude this section by showing that Average and q-Quantile are #P-hard under Turing reductions.

Theorem 5.6. *Let $0 < q < 1$ be a fixed number. The problems Average[VSA, $CWidth_{\mathbb{Q}^+}$] and q-Quantile[VSA, CWidth] are #P-hard under Turing reductions, even if w is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with*

$$R(d) := \begin{cases} 1 & \text{if } d = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Recall that Count[VSA] is #P-hard under Turing reductions (Corollary 3.2). We begin by giving a Turing reduction from Count[VSA] to Average[VSA, CWidth]. Let A, d, and d' be as defined in the proof of Theorem 5.5. The VSet-automaton A' builds upon A but selects a single additional tuple t with $\mathrm{t}(x) = [|d| + 2, |d| + 2\rangle$ for all variables. As we will see later, this tuple is used to calculate $\mathrm{Count}(\llbracket A \rrbracket, d)$ from $\mathrm{Avg}(\llbracket A' \rrbracket, d')$. Let $\mathsf{Vars}(A) = \{x_1, \ldots, x_n\}$. We define

$$A' := (A \cdot \triangleright_x 1 \triangleleft_x) \vee (d \cdot 1 \cdot \triangleright_{x_1} \triangleright_{x_2} \cdots \triangleright_{x_n} \triangleright_x \varepsilon \triangleleft_x \triangleleft_{x_n} \cdots \triangleleft_{x_2} \triangleleft_{x_1}) .$$

Observe that, for all $\mathrm{t} \in A'(d')$ it holds that $d_{\mathrm{t}(x)} = 1$ if and only if $\pi_{\mathsf{Vars}(A)}\mathrm{t} \in \llbracket A \rrbracket(d)$. Thus, per definition of A' and w, $\mathrm{Sum}(\llbracket A' \rrbracket, d', w) = \mathrm{Count}(\llbracket A \rrbracket, d)$ and $\mathrm{Count}(\llbracket A' \rrbracket, d') = \mathrm{Count}(\llbracket A \rrbracket, d) + 1$. Therefore, it holds that

$$\mathrm{Avg}(\llbracket A' \rrbracket, d', w) = \frac{\mathrm{Count}(\llbracket A \rrbracket, d)}{\mathrm{Count}(\llbracket A \rrbracket, d) + 1} .$$

Solving the equation for $\mathrm{Count}(\llbracket A \rrbracket, d)$, we have that

$$\mathrm{Count}(\llbracket A \rrbracket, d) = \frac{\mathrm{Avg}(\llbracket A' \rrbracket, d', w)}{1 - \mathrm{Avg}(\llbracket A' \rrbracket, d', w)} .$$

This concludes the proof that Average[VSA, $\mathrm{CWidth}_{\mathbb{Q}^+}$] is #P-hard under Turing reductions.

It remains to show that q-Quantile[VSA, CWidth] is also #P-hard under Turing reductions. Let $A \in \mathrm{VSA}$ be a VSet-automaton and $d \in \Sigma^*$ be a document. We will show the lower bound for $q = \frac{1}{2}$ first and study the general case of $0 < q < 1$ afterwards. Let $x \notin \mathsf{Vars}(A)$ be a new variable. Let $0 \leq r \leq |\mathsf{Spans}(d)|^{|\mathsf{Vars}(A)|}$. By Lemma 4.7 there is a VSet-automaton A' and a document d' with $\mathrm{Count}(\llbracket A' \rrbracket, d') = |\llbracket A' \rrbracket(d')| = r$. Let $0, 1 \notin \Sigma$ be a new alphabet symbol. Let $d_r = 0 \cdot d \cdot 1 \cdot d'$ and

$$A_r = \left(\triangleright_x \, 0 \triangleleft_x \cdot A \cdot 1 \cdot d' \right) \vee \left(0 \cdot d \cdot \triangleright_x 1 \triangleleft_x \cdot A' \right) .$$

Thus, $\mathrm{Count}(\llbracket A_r \rrbracket, d_r) = \mathrm{Count}(\llbracket A \rrbracket, d) + \mathrm{Count}(\llbracket A' \rrbracket, d')$. Recalling the definition of w it holds, for every tuple $\mathrm{t} \in \llbracket A_r \rrbracket$, that $w(d_r, \mathrm{t}) = 1$ if t was selected by A' and $w(d_r, \mathrm{t}) = 0$ otherwise, i.e., t was selected by A. Therefore, $\frac{1}{2}$-Quantile$(\llbracket A_r \rrbracket, d_r, w) = 0$ if and only if $\mathrm{Count}(\llbracket A \rrbracket, d) \geq \mathrm{Count}(\llbracket A' \rrbracket, d') = r$. Let $r_{\max}$ be the biggest r such that we have $\frac{1}{2}$-Quantile$(\llbracket A_r \rrbracket, d_r, w) = 0$. Using binary search, we can calculate $r_{\max}$ with a polynomial number of calls to an $\frac{1}{2}$-Quantile oracle. Furthermore, due to $\mathrm{Count}(\llbracket A \rrbracket, d) \in \mathbb{N}$ and $R_{\max}$ being maximal, it must hold that $\mathrm{Count}(\llbracket A \rrbracket, d) = r_{\max}$, concluding this part of the proof.

The general case of $0 < q < 1$ follows by slightly adopting the above reduction. Let $q = \frac{a}{b}$ with $a, b \in \mathbb{N}$ be given by its numerator and denominator. Observe that $b > a$ as $0 < \frac{a}{b} < 1$. Let A', d' be as above and let $c := \mathrm{Count}(\llbracket A \rrbracket, d)$. The document d_r consists of a copies of d, separated by $0's$ and $(b - a)$ copies of d' separated by $1's$. Formally,

$d_r = 0 \cdot d_1 \cdot 0 \cdot d_2 \cdot 0 \cdots d_a \cdot 0 \cdot 1 \cdot d'_1 \cdot 1 \cdot d'_2 \cdot 1 \cdots d'_{b-a} \cdot 1$, where each d_i (resp. d'_i) is a copy of d (resp. d'). Furthermore, let

$$A_r = \left(\Sigma_0^* \cdot \triangleright_x 0 \triangleleft_x \cdot A \cdot 0 \cdot \Sigma_0^* \cdot \Sigma_1^*\right) \vee \left(\Sigma_1^* \cdot \Sigma_1^* \cdot \triangleright_x 1 \triangleleft_x \cdot A' \cdot 1 \cdot \Sigma_1^*\right) ,$$

where $\Sigma_0 := \Sigma \cup \{0\}$ (resp. $\Sigma_1 := \Sigma \cup \{1\}$). Observe that w assigns 0 to exactly $c \cdot a$ tuples in $[\![A_r]\!](d_r)$ and $\mathrm{Count}([\![A_r]\!], d_r) = c \cdot a + r \cdot (b-a)$. Thus, $\frac{a}{b}$-Quantile$(A_r, d_r, w) = 0$ if and only if $\frac{c \cdot a}{c \cdot a + r \cdot (b-a)} \geq \frac{a}{b}$. We now show that $c \geq r$ if and only if $\frac{a}{b}$-Quantile$([\![A_r]\!], d_r, w) = 0$. Assume that $c \geq r$. Then,

$$\frac{c \cdot a}{c \cdot a + r \cdot (b-a)} \geq \frac{c \cdot a}{c \cdot a + c \cdot (b-a)} = \frac{c \cdot a}{c \cdot b} = \frac{a}{b} .$$

Therefore, $\frac{a}{b}$-Quantile$([\![A_r]\!], d_r, w) = 0$. On the other hand, if $c < r$,

$$\frac{c \cdot a}{c \cdot a + r \cdot (b-a)} < \frac{c \cdot a}{c \cdot a + c \cdot (b-a)} = \frac{c \cdot a}{c \cdot b} = \frac{a}{b} .$$

Thus, $\frac{a}{b}$-Quantile$([\![A_r]\!], d_r, w) = 1$.

Recall that $c = \mathrm{Count}([\![A]\!], d)$. As for $q = \frac{1}{2}$, let $r_{\max}$ be the biggest r such that $\frac{a}{b}$-Quantile$([\![A_r]\!], d_r, w) = 0$. Using binary search, we can calculate $r_{\max}$ with a polynomial number of calls to an $\frac{a}{b}$-QUANTILE oracle. Again it holds that $\mathrm{Count}([\![A]\!], d) = r_{\max}$, concluding the proof. □

6. Polynomial-Time Weight Functions

Before we study regular weight functions, we make a few observations on the very general polynomial-time computable weight functions. For weight functions $w \in$ POLY, we assume that w is represented as a Turing Machine A that returns a value $A(d, \mathrm{t})$ in polynomially many steps for some fixed polynomial of choice (e.g., n^2).[11] Furthermore, to avoid complexity due to the need to verify whether A is indeed a valid input (i.e., timely termination), we will assume that $w(d, \mathrm{t}) = 0$, if A does not produce a value within the allocated time.

We first observe that polynomial-time weight functions make all our aggregation problems intractable, which is not surprising. In fact, all the lower bounds already hold for regular weight functions.

Theorem 6.1. *The problems* MIN[uVSA, POLY] *and* MAX[uVSA, POLY] *are* OptP*-hard. Furthermore,* SUM[uVSA, POLY] *and* AVERAGE[uVSA, POLY] *are* #P*-hard.*

Proof. We will see later that these problems are already hard for weight functions in REG, which are a subclass of POLY (Theorems 7.3 and 7.7). □

Theorem 6.2. *Let* $0 < q < 1$. *Then* q*-*QUANTILE[uVSA, POLY] *is* #P*-hard under Turing reductions.*

Proof. We will see later that the problem is already hard for UREG weight functions (Theorem 7.9). □

We note that all studied problems can be solved in exponential time, by first constructing the relation $[\![A]\!](d)$, which might be of exponential size, computing the weights associated to all tuples, and finally computing the desired aggregate.

[11] Our complexity results are independent of the choice of this polynomial.

Theorem 6.3. *Let $0 < q < 1$. Then* Agg[VSA, Poly] *is in* FEXPTIME *for every* $\textsc{Agg} \in \{\textsc{Min}, \textsc{Max}, \textsc{Sum}, \textsc{Average}, q\text{-}\textsc{Quantile}\}$.

Proof. Let $A \in \text{VSA}$, $d \in \Sigma^*$, and $w \in \textsc{Poly}$. The algorithm first computes the multiset

$$W_{A,d,w} := \{\!\{ w(d, \mathrm{t}) \mid \mathrm{t} \in [\![A]\!](d) \}\!\} ,$$

which might be exponentially large. It is easy to see that $W_{A,d,w}$ can be computed in exponential time. Furthermore, it follows directly that Agg[VSA, Poly] is in FEXPTIME for every $\textsc{Agg} \in \{\textsc{Min}, \textsc{Max}, \textsc{Sum}, \textsc{Average}, q\text{-}\textsc{Quantile}\}$. □

Throughout this section, we do not study excessively whether we can give a more precise upper bound than the general FEXPTIME upper bound. However, we sometimes give such bounds. For instance, we are able to provide OptP and $\text{FP}^{\#\text{P}}$ upper bounds if the weight functions return natural numbers (or integers in the case of the $\text{FP}^{\#\text{P}}$ upper bounds).

Theorem 6.4. *Min*[VSA, *Poly*] *and* *Max*[VSA, *Poly*] *are in* OptP *if the weight function only assigns natural numbers.*

Proof. We only give the upper bound for Max. The proof for Min is analogous. To this end, let $A \in \text{VSA}$, $d \in \Sigma^*$, and $w \in \textsc{Poly}$ be a weight function which only assigns natural numbers. The Turing Machine N guesses a d-tuple t and accepts with output 0 if $\mathrm{t} \notin A(d)$. Otherwise, N computes the weight $w(d, \mathrm{t})$ and accepts with output $w(d, \mathrm{t})$. It is easy to see that the maximum output value of N is exactly $\text{Max}([\![A]\!], d, w)$. □

In the following theorem we show that Sum, Average, and q-Quantile can be computed in $\text{FP}^{\#\text{P}}$ if all weights are integers. The key idea is that, due to the restriction to integer weights, we can compute the aggregates by multiple calls to a #P oracle. For instance for Sum, we define two weight functions, w^+ and w^-, such that w^+ computes the sum of all positive and w^- the sum of all negative weights. Each of these sums can be computed by a single call to a #P oracle.

Theorem 6.5. *For every $0 \leq q \leq 1$, the problems* *Sum*[VSA, *Poly*], *Average*[VSA, *Poly*], *and* q-*Quantile*[VSA, *Poly*] *are in* $\text{FP}^{\#\text{P}}$ *if the weight function only assigns integers.*

Proof. We first prove that Sum[VSA, Poly] is in #P if the weight function only assigns natural numbers. We will use this as an oracle for the general upper bound. Let A be a VSet-automaton, $d \in \Sigma^*$ be a document and $w \in \textsc{Poly}$ be a weight function that only assigns natural numbers. A counting Turing Machine M for solving the problem in #P would have $w(d, \mathrm{t})$ accepting runs for every tuple in $A(d)$. More precisely, M guesses a d-tuple t over $\mathsf{Vars}(A)$ and checks whether $\mathrm{t} \in [\![A]\!](d)$. If $\mathrm{t} \in [\![A]\!](d)$ and $w(d, \mathrm{t}) > 0$, then M branches into $w(d, \mathrm{t})$ accepting branches, which it can do because w is given in the input as a polynomial-time deterministic Turing Machine. Otherwise, M rejects. Per construction, M has exactly $w(d, \mathrm{t})$ accepting branches for every tuple $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) > 0$. Thus, the number of accepting runs is exactly $\sum_{\mathrm{t} \in [\![A]\!](d)} w(d, \mathrm{t}) = \text{Sum}([\![A]\!], d, w)$.

We now continue by showing that Sum[VSA, Poly] is in $\text{FP}^{\#\text{P}}$ if the weight function only assigns integers. Let A be a VSet-automaton, $d \in \Sigma^*$ be a document, and $w \in \textsc{Poly}$ be a weight function, which only assigns integers.

We define two weight functions $w^+, w^- \in \textsc{Poly}$, such that

$$\text{Sum}(A, d, w) = \text{Sum}(A, d, w^+) - \text{Sum}(A, d, w^-) .$$

Formally, we define the following two weight functions:

$$w^+(d,\mathrm{t}) := \begin{cases} w(d,\mathrm{t}) & \text{if } w(d,\mathrm{t}) \geq 0, \text{ and} \\ 0 & \text{otherwise;} \end{cases}$$

$$w^-(d,\mathrm{t}) := \begin{cases} -w(d,\mathrm{t}) & \text{if } w(d,\mathrm{t}) < 0, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, $\mathrm{Sum}(\llbracket A \rrbracket, d, w) = \mathrm{Sum}(\llbracket A \rrbracket, d, w^+) - \mathrm{Sum}(\llbracket A \rrbracket, d, w^-)$ and the answer to $\textsc{Sum}[\mathcal{S}, \textsc{Poly}]$ can be obtained by taking the difference of the answers of two calls to the $\textsc{Sum}[\mathcal{S}, \textsc{Poly}]$ #P oracle. The upper bound for $\textsc{Average}[\mathrm{VSA}, \textsc{Poly}]$ is immediate from the upper bound of $\textsc{Sum}[\mathrm{VSA}, \textsc{Poly}]$ and Theorem 3.1. For the upper bound of q-$\textsc{Quantile}[\mathrm{VSA}, \textsc{Poly}]$ we define the weight function

$$w_{\leq k}(d,\mathrm{t}) = \begin{cases} 1 & \text{if } w(d,\mathrm{t}) \leq k, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Recall that

$$q\text{-Quantile}(S,d,w) := \min \left\{ r \in \mathrm{Img}(S,d,w) \;\middle|\; \frac{|\{\mathrm{t} \in S(d) \mid w(d,\mathrm{t}) \leq r\}|}{|S(d)|} \geq q \right\}.$$

And therefore

$$q\text{-Quantile}(S,d,w) = \min \left\{ r \in \mathrm{Img}(S,d,w) \;\middle|\; \frac{\mathrm{Sum}(\llbracket A \rrbracket, d, w_{\leq k})}{\mathrm{Count}(\llbracket A \rrbracket, d)} \geq q \right\}.$$

Thus, the upper bound of q-$\textsc{Quantile}[\mathrm{VSA}, \textsc{Poly}]$ can be obtained by performing binary search, using the upper bound of $\textsc{Sum}[\mathrm{VSA}, \textsc{Poly}]$ and Theorem 3.1. □

7. Regular Weight Functions

We now turn to Reg and UReg weight functions. As we have shown in Proposition 4.3, every CWidth weight function can be translated into an equivalent UReg weight function. Furthermore, the weight functions which were used for the lower bounds can be represented by unambiguous weighted VSet-automata of constant size. Therefore, all lower bounds for CWidth also hold for UReg.

7.1. Compact DAG Representation.

As we show next, aggregation problems for regular weight functions can often be reduced to problems about paths on weighted *directed acyclic graphs (DAGs)*, where the weights come from the semiring of the weight function. To this end, let $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring. A $\mathbb{K}$-*weighted DAG* is a DAG $D = (N, E)$, where N is a set of nodes, $E \subseteq N \times \mathbb{K} \times N$ is a finite set of weighted edges, and *src* (resp., *snk*) is a unique node in N without incoming (resp., outgoing) edges. We define $\mathrm{len}(e) = \ell$, where $e = (v, \ell, v') \in E$. Furthermore, we define paths p in the obvious manner as sequences of edges and the length $\mathrm{len}(p)$ of p as the product ($\otimes$) of the lengths of its edges. More formally, a path

$$p := n_1 \ell_1 n_2 \cdots \ell_{n-1} n_j$$

is a sequence of nodes $n_i \in N$ with $1 \leq i \leq j$ and $(n_i, \ell_i, n_{i+1},) \in E$, for all $1 \leq i < j$, and the length

$$\mathrm{len}(p) := \ell_1 \otimes \cdots \otimes \ell_{j-1}\,.$$

We denote the set of all paths in D from src to snk by $\mathrm{Paths}(src, snk)$.

Given a document d, a VSet-automaton A and a regular weight function $w \in \textsc{Reg}_{\mathbb{K}}$, we will construct a DAG D which plays the role of a compact representation of the materialized intermediate result. The DAG D is obtained by a product construction between A, W, and d, such that every path from src to snk corresponds to an accepting run of W that represents a tuple in $[\![A]\!](d)$. If A and W are unambiguous this correspondence is actually a bijection.

Lemma 7.1. *Let $\mathbb{K} \in \{\mathbb{Q}, \mathbb{T}\}$ be either the numerical or the tropical semiring. Let d be a document, $A \in$ VSA, and W be the weighted VSet-automaton representing $w \in \textsc{Reg}_{\mathbb{K}}$. We can compute, in polynomial time, a $\mathbb{K}$-weighted DAG D, such that there is a surjective mapping m from paths $p \in Paths(src, snk)$ in D to tuples $\mathrm{t} \in [\![A]\!](d)$. Furthermore,*

(1) *the mapping m is a bijection, if A and W are unambiguous, and*
(2) $w(d, \mathrm{t}) = \bigoplus_{p \in Paths(src,snk), m(p)=\mathrm{t}} len(p)$, *for every $\mathrm{t} \in [\![A]\!](d)$, if $A \in$ uVSA or $\mathbb{K} = \mathbb{T}$.*

Proof. Let $d \in \Sigma^*$, $A \in$ VSA, and W be the weighted VSet-automaton representing $w \in \textsc{Reg}_{\mathbb{K}}$. By Proposition 4.1, we can assume, w.l.o.g., that all VSet-automata used in this proof do not contain ε-transitions.

We begin by giving the construction of D. Let W_A be the weighted VSet-automaton obtained by interpreting A as a $\mathbb{K}$-weighted VSet-automaton. More formally, every transition in A is interpreted as a weighted transition with weight $\overline{1}$ and every transition which is not in A is interpreted as a transition with weight $\overline{0}$. Furthermore, let $W_d := d$ be the weighted VSet-automaton with $\mathsf{Vars}(W_d) = \emptyset$ that assigns the weight $\overline{1}$ to the empty tuple on input d and $\overline{0}$ to every tuple on input $d' \neq d$. By Theorem 4.2 the join of weighted VSet-automata can be computed in polynomial time. Let

$$W_D := W \bowtie W_A \bowtie W_d \ .$$

Per definition of join for $\mathbb{K}$-relations, it holds that

$$[\![W_D]\!]_{\mathbb{K}}(d, \mathrm{t}) = [\![W]\!]_{\mathbb{K}}(d, \pi_{\mathsf{Vars}(W)}(\mathrm{t})) \otimes [\![W_A]\!]_{\mathbb{K}}(d, \pi_{\mathsf{Vars}(W_A)}(\mathrm{t})) \otimes [\![W_d]\!]_{\mathbb{K}}(d, \pi_{\mathsf{Vars}(W_d)}(\mathrm{t})) \ .$$

Let $A \in$ uVSA be unambiguous or $\mathbb{K} = \mathbb{T}$. In both cases, it holds that

$$[\![W_A]\!]_{\mathbb{K}}(d, \mathrm{t}) = \begin{cases} \overline{1} & \text{if } \mathrm{t} \in [\![A]\!](d), \text{ and} \\ \overline{0} & \text{otherwise.} \end{cases}$$

Furthermore,

$$[\![W_d]\!]_{\mathbb{K}}(d', \mathrm{t}) = \begin{cases} \overline{1} & \text{if } \mathsf{Vars}(\mathrm{t}) = \emptyset \text{ and } d' = d, \text{ and} \\ \overline{0} & \text{otherwise.} \end{cases}$$

Therefore, if $A \in$ uVSA or $\mathbb{K} = \mathbb{T}$, it holds, for every tuple $\mathrm{t} \in [\![A]\!](d)$. that

$$[\![W_D]\!]_{\mathbb{K}}(d, \mathrm{t}) = [\![W]\!]_{\mathbb{K}}(d, \pi_{\mathsf{Vars}(W)}(\mathrm{t})) \quad (\dagger)$$

We will use this equality in the proof of condition (2).

The DAG $D = (N_D, E_D)$ is obtained from $W_D = (\Sigma, V, Q, I, F, \delta)$ as follows. The set of nodes $N_D := \big(Q \times (\Sigma \cup \Gamma_V \cup \emptyset)\big) \uplus \{src, snk\}$ contains the nodes src, snk, plus a state (q, σ) for each $q \in Q$ and $\sigma \in (\Sigma \cup \Gamma_V \cup \emptyset)$, where $\sigma \neq \emptyset$ encodes the label of the last transition

and q the state. The set of edges is defined as follows:

$$\begin{aligned} E_D :=& \{(src, \ell, (x, \emptyset)) \mid I(x) = \ell \neq \infty\} \\ & \uplus \{((x_1, \sigma_1), \ell, (x_2, \sigma_2) \mid \delta(x_1, \sigma_2, x_2) = \ell \neq \overline{0}, \text{ where } \sigma_1 \in (\Sigma \cup \Gamma_V \cup \emptyset)\} \\ & \uplus \{((x, \sigma), \ell, snk) \mid F(x) = \ell \neq \infty, \text{ where } \sigma \in (\Sigma \cup \Gamma_V \cup \emptyset)\} \ . \end{aligned}$$

In the following we assume that D is trimmed, that is, for every node $n \in N_D$ there is at least one path from src to snk, which visits n.[12]

We observe that the construction of D only requires polynomial time. Note that there is a one-to-one correspondence between paths $p \in \text{Paths}(src, snk)$ and accepting runs of W_D on d. That is,

$$p = src \cdot \ell_0 \cdot (q_0, \emptyset) \cdot \ell_1 \cdot (q_1, \sigma_1) \cdots (q_n, \sigma_n) \cdot \ell_{n+1} \cdot snk$$

is a path from src to snk in D if and only if

$$\rho = q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} q_n \ ,$$

with $I(q_0) = \ell_0$ and $F(q_n) = \ell_{n+1}$ is an accepting run of W_D on d. Furthermore, we observe that the weight of p is exactly the weight assigned to the run ρ by W_D, that is, $\text{len}(p) = \mathsf{w}_\rho$.

For the sake of contradiction, assume that D is cyclic. Per assumption, all nodes $n \in N$ are on a path from src to snk, thus, D must have a path p from src to snk, which contains a cycle. Let ρ be the run of W_D corresponding to p. The automaton W_d is acyclic. Observe that W_D is functional as W, W_A, and W_d are functional. Thus, $\text{ref}(\rho)$ is valid and therefore the cycle can not contain an edge labeled by a variable operation. Per assumption, all involved VSet-automata do not contain ε-transitions. Therefore, the cycle must only consist of edges, labeled by alphabet symbols. Let ρ' be the run, obtained from ρ by removing all cycles. Due to commutativity of $\otimes$, it follows that $\mathsf{w}_{\rho'} = \mathsf{w}_\rho \otimes x$ for some $x \neq \overline{0}$. We observe that $\text{doc}(\text{ref}(\rho')) \neq d$. Therefore, there is a run ρ' of W_D on $\text{doc}(\text{ref}(\rho')) \neq d$ with weight $\mathsf{w}_{\rho'} \neq \overline{0}$, which is the desired contradiction to the observation that for all runs ρ of W_D it holds that $\mathsf{w}_\rho \neq \overline{0}$ if and only if $\text{doc}(\text{ref}(\rho)) = d$.

We now define the mapping m. Let $p \in \text{Paths}(src, snk)$ and let ρ be the corresponding run of W_D. We define the mapping $m(p) := \text{tup}(\rho)$. It follows directly that m is surjective. If $A \in \text{uVSA}$ or $\mathbb{K} = \mathbb{T}$ and for $\mathrm{t} \in [\![A]\!](d)$, we have that

$$\begin{aligned} w(d, \mathrm{t}) = \ & [\![W]\!]_{\mathbb{K}}(d, \pi_{\mathsf{Vars}(W)}(\mathrm{t})) \overset{(\dagger)}{=} [\![W_D]\!]_{\mathbb{K}}(d, \mathrm{t}) \\ = & \bigoplus_{\rho \in P(W_D, d) \text{ and } \mathrm{t} = \text{tup}(\rho)} \mathsf{w}_\rho = \bigoplus_{p \in \text{Paths}(src, snk), m(p) = \mathrm{t}} \text{len}(p) \ . \end{aligned}$$

The first and the third equalities follow from the definitions of Reg weight functions and $\mathbb{K}$-annotators. The last equality follows from the definition of D. This concludes the proof of condition (2).

It remains to show that condition (1) holds. Assume that $A \in \text{uVSA}$ and W are unambiguous. Then, by Theorem 4.2, W_D is unambiguous.[13] Assume that there are two paths $p_1 \neq p_2$ such that $p_1, p_2 \in \text{Paths}(src, snk)$ with $m(p_1) = m(p_2)$. Let $\rho_1 \neq \rho_2$ be the corresponding runs of W_D. Due to $m(p) = \text{tup}(\rho)$, it must hold that ρ_1 and ρ_2 are two runs

[12] Note that this condition can be enforced in linear time by two graph traversals (e.g. using breadth first search), one starting from src to identify all states which can be reached from src and one starting from snk to identify all states which can reach snk. We remove all states which are not marked by both graph traversals.

[13] Recall that W_d is unambiguous.

of W_D, encoding the same tuple t. Due to the unambiguity condition (C2) in Section 2.5, both runs must encode a different ref-word, that is, $\mathrm{ref}(\rho_1) \neq \mathrm{ref}(\rho_2)$ however this implies that either $\mathrm{ref}(\rho_1)$ or $\mathrm{ref}(\rho_2)$ must violate the variable order condition, contradicting the unambiguity condition (C1) in Section 2.5. Thus, m must be a bijection. □

7.2. **Min and Max Aggregation.** We will now study the computational complexity of Min and Max aggregation. We begin by giving the tractable cases which are based on Lemma 7.1. The weighted DAG from Lemma 7.1 allows us to reduce Min to the shortest path problem in DAGs. If the weight function is unambiguous, Max can be reduced to the longest path problem in DAGs. Notice that, although the longest path problem is intractable in general, it is tractable for DAGs.

Theorem 7.2. *The problems* $\mathit{Min}[\mathrm{VSA}, \mathit{Reg}_{\mathbb{T}}]$, $\mathit{Min}[\mathrm{uVSA}, \mathit{UReg}_{\mathbb{Q}}]$, $\mathit{Max}[\mathrm{VSA}, \mathit{UReg}_{\mathbb{T}}]$, *and* $\mathit{Max}[\mathrm{uVSA}, \mathit{UReg}_{\mathbb{Q}}]$ *are in* FP.

Proof. Let d be a document, $A \in \mathrm{VSA}$, and W be the weighted VSet-automaton representing $w \in \mathrm{Reg}_{\mathbb{T}}$ or $w \in \mathrm{UReg}_{\mathbb{Q}}$. Let D and m be the DAG and the surjective mapping as guaranteed by Lemma 7.1. In the following, we will reduce all four cases to finding the path with minimal (resp., maximal) length in D. Note that given a weighted DAG D, one can compute the path with minimal (resp., maximal) length in polynomial time, via dynamic programming, e.g. using the Bellman-Ford algorithm.[14]

We begin by giving the proofs for the numerical semiring. If $A \in \mathrm{uVSA}$ and $W \in \mathrm{UReg}_{\mathbb{Q}}$, it follows directly from property (1) of Lemma 7.1 that m is a bijection. Therefore, for every tuple $\mathrm{t} \in [\![A]\!](d)$, there is exactly one path $p \in \mathrm{Paths}(\mathit{src}, \mathit{snk})$ with $m(p) = \mathrm{t}$. Thus, $w(d, \mathrm{t}) = \mathrm{len}(p)$, where $p \in \mathrm{Paths}(\mathit{src}, \mathit{snk})$ with $m(p) = \mathrm{t}$. It follows directly that $\mathrm{Min}([\![A]\!], d, w)$ and $\mathrm{Max}([\![A]\!], d, w)$ can be computed from D by searching for the path p with minimal (respectively maximal) length.

It remains to give the proofs for the tropical semiring. We begin by giving the proof for $\mathrm{Min}[\mathrm{VSA}, \mathrm{Reg}_{\mathbb{T}}]$. Due to property (2) of Lemma 7.1,

$$\mathrm{Min}([\![A]\!], d, w) = \min_{\mathrm{t} \in [\![A]\!](d)} \min_{p \in \mathrm{Paths}(\mathit{src}, \mathit{snk}), m(p) = \mathrm{t}} \mathrm{len}(p) = \min_{p \in \mathrm{Paths}(\mathit{src}, \mathit{snk})} \mathrm{len}(p)$$

and therefore $\mathrm{Min}[\mathrm{VSA}, \mathrm{Reg}_{\mathbb{T}}]$ again reduces to computing the path of minimal length in D.

For Max, the situation is different, because the maximal weight of an output tuple is

$$\mathrm{Max}([\![A]\!], d, w) = \max_{\mathrm{t} \in [\![A]\!](d)} \min_{p \in \mathrm{Paths}(\mathit{src}, \mathit{snk}), m(p) = \mathrm{t}} \mathrm{len}(p) \ .$$

However, if W is unambiguous, it must hold that $\mathrm{len}(p) = \mathrm{len}(p')$ for all runs $p, p' \in \mathrm{Paths}(\mathit{src}, \mathit{snk})$ with $m(p) = m(p')$. Otherwise W would be required to have at least two runs which accept the same tuple but assign different weights. Thus, W would not be unambiguous. We can therefore conclude that,

$$\mathrm{Max}(S, d, w) = \max_{\mathrm{t} \in [\![A]\!](d)} \min_{\{p | m(p) = \mathrm{t}\}} \mathrm{len}(p) = \max_{p \in \mathrm{Paths}(\mathit{src}, \mathit{snk})} \mathrm{len}(p) \ .$$

Again, we can reduce $\mathrm{Max}[\mathrm{VSA}, \mathrm{UReg}_{\mathbb{T}}]$ to the max length problem on D. □

[14] One has to be careful in the case of the numeric semiring as the lengths along the path are multiplied. Therefore one has to maintain the minimal as well as the maximal length between two nodes, as edges with negative length change the sign, resulting in minimal path's to be maximal and vice versa.

As we show now, the results of Theorem 7.2 are close to the tractability frontier: For instance, if we relax the unambiguity condition in the weight function, the problem MAX does not correspond to finding the longest paths in DAGs and becomes intractable.

Theorem 7.3. *MIN*[uVSA, $\mathrm{REG}_{\mathbb{Q}}$], *MAX*[uVSA, $\mathrm{REG}_{\mathbb{T}}$], *and MAX*[uVSA, $\mathrm{REG}_{\mathbb{Q}}$] *are* OptP*-hard.*

Proof. We begin by giving the proofs for MAX[uVSA, $\mathrm{REG}_{\mathbb{T}}$]. We give a metric reduction[15] from the OptP-complete problem Maximum Satisfying Assignment (MSA) [Kre88], which is defined as follows. Let $\phi(x_1, \ldots, x_n)$ be a propositional formula in CNF and let $v = v_1 \cdots v_n \in \mathbb{B}^n$ be a variable assignment of ϕ. Furthermore, let $n_v \in \mathbb{N}$ be the natural number encoded by v in binary. MSA asks, given the CNF formula $\phi(x_1, \ldots, x_n)$, for the maximum $n_v \in \mathbb{N}$ such that v satisfies ϕ, or 0 if ϕ is not satisfiable. In the following, we denote by $\mathrm{MSA}(\phi)$ the output of MSA on input ϕ.

Let $\phi(x_1, \ldots, x_n)$ be a Boolean formula in CNF. We use a similar construction as in the proofs of Theorem 5.4 and Doleschal et al. [DKMP22, Theorem 7.6], to encode the CNF formula ϕ. Let $d = a^n$ be the document. We define

$$A := ((\triangleright_{x_1} \varepsilon \triangleleft_{x_1} a) \vee (\triangleright_{x_1} a \triangleleft_{x_1})) \cdots ((\triangleright_{x_n} \varepsilon \triangleleft_{x_n} a) \vee (\triangleright_{x_n} a \triangleleft_{x_n})) \,.$$

Notice that A can be defined with a polynomial-time constructible uVSA. Observe that there is a one-to-one correspondence between tuples t in $[\![A]\!](d)$ and variable assignments α_{t} for ϕ: we can set $\alpha_{\mathrm{t}}(x_i) = 1$ if and only if $\mathrm{t}(x_i) = [i, i+1\rangle$. We construct a weight function $w \in \mathrm{REG}_{\mathbb{T}}$ such that

$$w(d, \mathrm{t}) = \begin{cases} n_{\alpha_{\mathrm{t}}} & \text{if } \alpha_{\mathrm{t}} \models \phi \\ 0 & \text{otherwise.} \end{cases}$$

Recall that $n_{\alpha_{\mathrm{t}}}$ is the natural number which is encoded by the variable assignment α_{t}. It follows directly that $\mathrm{MSA}(\phi) = \mathrm{Max}([\![A]\!], d, w)$. Defining $T_2(x, y) \mapsto y$ gives the desired reduction.

It remains to construct a weighted VSet-automaton W which encodes w. We define the weighted VSet-automaton W as the union of two automata. Let V be the set of variables of ϕ. The first automaton W_A is a copy of A, assigning weight 0 to all edges, which are present in A. Furthermore, let δ assign weight 2^{i-1} to the a labeled edge between opening and closing variable x_i (that is, $\triangleright_{x_i}$ and $\triangleleft_{x_i}$). Let $I(q) = 0$ if q is the start state of A and ∞, otherwise. Analogously, let $F(q) = 0$ if q is an accepting state of A and ∞ otherwise. It follows directly that $[\![W_A]\!]_{\mathbb{K}}(a^n, \mathrm{t}) = n_{\alpha_{\mathrm{t}}}$.

The second automaton, W' consists of m disjoint branches, where each branch corresponds to a clause C_i of ϕ; we call these *clause branches*. Each branch has exactly one run ρ with weight $\overline{1}$ for each tuple t associated to an assignment α_{t} which does not satisfy the clause C_i.

We now give a formal construction of W'. The set of states $Q := \{q^a_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq a \leq 5\}$ contains $5n$ states for each clause branch. Intuitively, W' has a gadget, consisting of 5 states, for each variable and each clause branch. Figure 6 depicts the three types of gadgets we use here. Note that the weights of the drawn edges are all 0. We use the left gadget if x does not occur in the relevant clause and the middle (resp., right) gadget if the literal $\neg x$ (resp., x) occurs. Furthermore, within the same branch of W', the last state of each gadget is the same state as the start state of the next variable, i.e., $q^5_{i,j} = q^1_{i,j+1}$ for all $1 \leq i \leq k, 1 \leq j < n$.

[15] Recall that a metric reduction from f to g is a pair of polynomial-time computable functions T_1, T_2, where $T_1 : \Sigma^* \to \Sigma^*$ and $T_2 : \Sigma^* \times \mathbb{N} \to \mathbb{N}$, such that $f(x) = T_2(x, g(T_1(x)))$ for all $x \in \Sigma^*$.

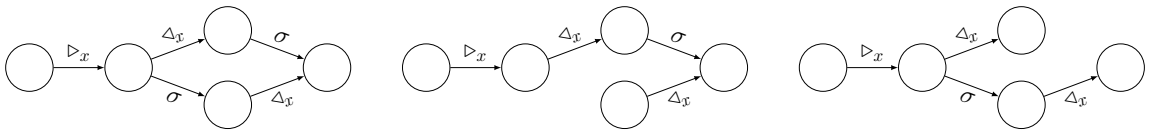

Figure 6: Example gadgets for variable x.

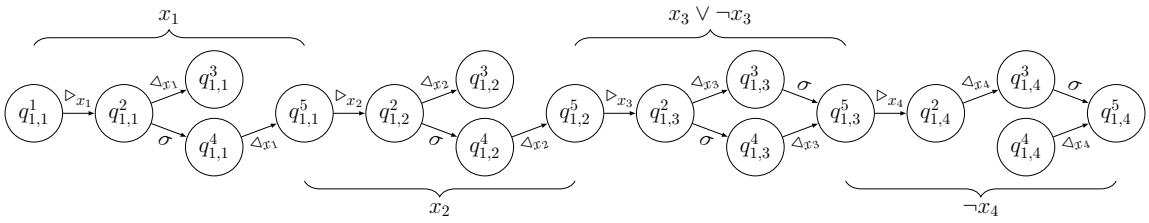

Figure 7: The clause branch of W corresponding to C_1 and $x_1 = x_2 = 1, x_4 = 0$.

We illustrate the crucial part of the construction on an example. Let $\phi = (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$. The corresponding weighted VSet-automaton W' therefore has two disjoint branches, one for each clause of ϕ. Figure 7 depicts the clause branch C_1 that corresponds to all assignments which do not satisfy C_i, that is, all assignments with $x_1 = x_2 = 1$ and $x_4 = 0$.

Formally, the initial weight function is $I(q^a_{i,j}) = \overline{1}$ if $j = 1 = a$ and $I(q^a_{i,j}) = \overline{0}$ otherwise. The final weight function $F(q^a_{i,j}) = \overline{1}$ if $j = n$ and $a = 5$ and $F(q^a_{i,j}) = \overline{0}$, otherwise. The transition function δ is defined as follows:

$$\delta(q^a_{i,j}, o, q^{a'}_{i,j}) = \begin{cases} \overline{1} & a = 1, a' = 2, o = \triangleright_{x_j} \\ \overline{1} & a = 2, a' = 3, o = \triangleleft_{x_j} \\ \overline{1} & a = 2, a' = 4, \text{ and there is a variable assignment } \tau \text{ with} \\ & \quad \tau(x_j) = 1 \text{ and } \tau \not\models C_i \\ \overline{1} & a = 3, a' = 5, o = a, \text{ and there is a variable assignment } \tau \text{ with} \\ & \quad \tau(x_j) = 0 \text{ and } \tau \not\models C_i \\ \overline{1} & a = 4, a' = 5, o = \triangleleft_{x_j} \end{cases}$$

All other transitions have weight $\overline{0}$.

We claim that W' represents w', where $w'(d, \mathrm{t}) = \overline{1}$ if $\alpha_{\mathrm{t}} \not\models \phi$ and $w'(d, \mathrm{t}) = \overline{0}$ otherwise. To this end, let $\mathrm{t} \in [\![A]\!](d)$ be a tuple and let $\tau = \alpha_{\mathrm{t}}$ be the variable assignment encoded by t. It is easy to see that there is an accepting run ρ of W' for r with weight $\mathsf{w}_\rho = \overline{1}$, starting in $q^a_{i,0}$, if and only if τ does not satisfy clause C_i.

As mentioned before, the weighted VSet-automaton W is the union of W' and W_A. Recall that, over the tropical semiring, $\overline{0} = \infty, \overline{1} = 0$, and the weight of a tuple t is the minimal weight over all accepting runs which encode t. Thus, the weight function represented by W is exactly w, as claimed. This concludes the proof that $\textsc{Max}[\text{uVSA}, \textsc{Reg}_{\mathbb{T}}]$ is OptP-hard.

It remains to show that $\textsc{Min}[\text{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$ and $\textsc{Max}[\text{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$ are OptP-hard. We first show OptP-hardness for $\textsc{Max}[\text{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$.

We give a metric reduction from the OptP-complete problem of weighted satisfiability (WSAT) [Kre88], which is defined as follows. Let $\phi(x_1, \ldots, x_n)$ be a propositional formula in CNF with binary weights. WSAT asks, given the CNF formula $\phi(x_1, \ldots, x_n)$ with m clauses

and weights $w_1, \ldots, w_m$, for the maximal weight of an assignment, where the weight of an assignment is the sum of the weights of the satisfied clauses.

Denote by $\mathrm{WSAT}(\phi)$ the output of WSAT on input ϕ. Let $\phi(x_1, \ldots, x_n)$ be a Boolean formula in CNF. Let d, A, W be as defined before. However, the weights in W are defined differently. That is, W is the union of W_A and W', where W_A is a copy of A, where all transitions have weight $\overline{1}$. Furthermore, let x be the sum of all clause weights and $F(q) = x$, if q is an accepting state of A. The automaton W' is defined exactly as before, however, accepting with final weight $F(q) = -w_i$ if q is the final weight of the branch of clause C_i and w_i is the weight of C_i. Observe that $w(d, \mathrm{t}) = [\![W]\!]_{\mathbb{Q}}(d, \mathrm{t})$ is exactly the weighted sum of all clauses, which are satisfied by the valuation α_{t} encoded by t. It follows that $\mathrm{Max}(S, d, w) = \mathrm{WSAT}(\phi)$. Defining $T_2(x, y) \mapsto y$ concludes the proof for $\textsc{Max}[\mathrm{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$.

The proof for $\textsc{Min}[\mathrm{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$ is analogous, replacing the weight x with $-x$ and $-w_i$ with weight w_i. Therefore, $\mathrm{Min}(S, d, w) = -\mathrm{WSAT}(\phi)$. Defining $T_2(x, y) \mapsto -y$ concludes the proof. □

7.3. **Sum and Average Aggregation.** Since Sum and Average are already intractable for VSA spanners and CWidth weight functions (Theorems 5.4, 5.5, and 5.6), they are intractable for VSA spanners and Reg/UReg weight functions as well. In a similar vein as in Section 5, the problems become tractable if we have unambiguity. However, in the case of the tropical semiring, we require unambiguity of *both* the spanner and the representation of the weight function. We begin by showing that $\textsc{Sum}[\mathrm{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$ and $\textsc{Sum}[\mathrm{uVSA}, \textsc{UReg}_{\mathbb{T}}]$ are in FP. In both cases the problem can be reduced to computing the sum of the lengths of source-to-target paths in a DAG, by using Lemma 7.1.

Theorem 7.4. *$\textsc{Sum}[\mathrm{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$ is in* FP.

Proof. Let $d \in \Sigma^*$, $A \in \mathrm{uVSA}$, and W be a weighted VSet-automaton representing $w \in \textsc{Reg}_{\mathbb{Q}}$. Let $D = (N, E)$ and m be as guaranteed by Lemma 7.1. It follows that

$$\mathrm{Sum}([\![A]\!], d, w) = \sum_{\mathrm{t} \in [\![A]\!](d)} \sum_{p \in \mathrm{Paths}(src, snk), m(p) = \mathrm{t}} \mathrm{len}(p) = \sum_{p \in \mathrm{Paths}(src, snk)} \mathrm{len}(p) \ .$$

All paths $p \in \mathrm{Paths}(src, snk)$ consist of $|d| + 1 + 2 \cdot |\mathsf{Vars}(A)|$ edges. We assume, w.l.o.g., that $N = \{1, \ldots, n\}$, with $src = 1$ and $snk = n$. Therefore, $\mathrm{Sum}([\![A]\!], d, w)$ can be computed by interpreting the edge relation E as a $\mathbb{Q}^{n \times n}$ matrix M and computing the weight

$$I \times M^{|d| + 1 + 2 \cdot |\mathsf{Vars}(A)|} \times F^T \ ,$$

where $I = (1, 0, \ldots, 0)$ (resp., $F = (0, \ldots, 0, 1)$) is the vector which assigns 0 to all nodes but 1 (resp., n), which is assigned the weight 1. Recall that the numerical semiring has an efficient encoding. Therefore, $\mathrm{Sum}(S, d, w)$ can indeed be computed in polynomial time. □

Theorem 7.5. *$\textsc{Sum}[\mathrm{uVSA}, \textsc{UReg}_{\mathbb{T}}]$ is in* FP.

Proof. Let D, m be the DAG and the bijection guaranteed by Lemma 7.1. We have that

$$\begin{aligned}\mathrm{Sum}(\llbracket A \rrbracket, d, w) &\overset{(1)}{=} \sum_{\mathrm{t} \in \llbracket A \rrbracket(d)} w(d, \mathrm{t}) \\ &\overset{(2)}{=} \sum_{\mathrm{t} \in \llbracket A \rrbracket(d)} \min_{p \in \mathrm{Paths}(src, snk), m(p) = \mathrm{t}} \mathrm{len}(p) \\ &\overset{(3)}{=} \sum_{p \in \mathrm{Paths}(src, snk)} \mathrm{len}(p) \ .\end{aligned}$$

The first equation follows from the definition of Sum. The second equation follows from property (2) of Lemma 7.1. The third equation must hold due to m being a bijection between tuples $\mathrm{t} \in \llbracket A \rrbracket$ and paths $p \in \mathrm{Paths}(src, snk)$.

It remains to show that the sum of the lengths of source-to-target paths in a DAG $D = (N, E)$ can be computed in polynomial time. We begin by observing that given two nodes $x, y \in D$ the number of paths from x to y in D can be computed in polynomial time via dynamic programming. Furthermore, given an edge $e = (x, y) \in E$ one can compute the number of paths from src to snk which use e by multiplying the number of path's from src to x with the number of paths from y to snk. Therefore, the function $c : E \to \mathbb{N}$ which, given an edge $e \in E$ assigns the number of paths using e can be computed in polynomial time. Recall that over the tropical semiring, $\otimes = +$ and therefore $\mathrm{len}(p) = \sum_{e \in p} \mathrm{len}(e)$. It therefore follows that

$$\begin{aligned}\mathrm{Sum}(\llbracket A \rrbracket, d, w) &= \sum_{p \in \mathrm{Paths}(src, snk)} \mathrm{len}(p) \\ &= \sum_{p \in \mathrm{Paths}(src, snk)} \sum_{e \in p} \mathrm{len}(e) \\ &= \sum_{e \in E} (\mathrm{len}(e) \times c(e)) \ .\end{aligned}$$

Therefore, Sum can be computed by representing the weights $\mathrm{len}(e)$ as a vector I and the counts $c(e)$ as a vector F. Thus, $\mathrm{Sum}(\llbracket A \rrbracket, d, w) = I \times F^T$, which can be computed in polynomial time, as $\mathrm{Reg}_{\mathbb{T}}$ has an efficient encoding. □

We observe that FP upper bounds for Average follows directly from the corresponding upper bound for Sum and the FP upper bound for Count (Theorem 3.1).

Corollary 7.6. *Average*[uVSA, $Reg_{\mathbb{Q}}$] *and* *Average*[uVSA, $UReg_{\mathbb{T}}$] *are in* FP.

If we relax the restriction that weight functions are given as unambiguous automata, Sum and Average become #P-hard again.

Theorem 7.7. *Sum*[uVSA, $Reg_{\mathbb{T}}$] *and* *Average*[uVSA, $Reg_{\mathbb{T}}$] *are* #P*-hard.*

Proof. We begin by giving a parsimonious reduction from the #P-complete problem of #CNF. To this end, let $c = 1$ in the case of Sum and $c = 2^n$ in the case of Average.

Let $\phi(x_1, \ldots, x_n)$ be a propositional formula in conjunctive normal form. Let A, d be as constructed in the proof of Theorem 7.3 and let w be the weight function such that $w(d, \mathrm{t}) = c$ if the corresponding assignment α_{t} satisfies ϕ and $w(d, \mathrm{t}) = 0$ otherwise. Therefore, with $c := 1$

it follows directly that $\#\mathrm{CNF}(\phi) = \mathrm{Sum}(\llbracket A \rrbracket, d, w)$, which shows that the problem is #P-hard. For Average let $c := 2^n$. It follows that $\#\mathrm{CNF}(\phi) = x = \frac{x \cdot 2^n}{2^n} = \frac{x \cdot c}{2^n} = \mathrm{Avg}(\llbracket A \rrbracket, d, w)$, implying that Average[uVSA, Reg$_{\mathbb{T}}$] is also #P-hard.

It remains to show that there is a weighed automaton W representing $w \in \mathrm{Reg}_{\mathbb{T}}$. As in the proof of Theorem 7.3, W is the union of two weighted VSet-automata W_A and W', where W_A is a copy of A, assigning weight 0 to all initial states and transitions of A and weight c to all final states. Furthermore, W' is as defined, that is

$$\llbracket W' \rrbracket_{\mathbb{T}}(a^n, \mathrm{t}) = \begin{cases} 0 & \text{if } \alpha_{\mathrm{t}} \not\models \phi \\ \infty & \text{otherwise.} \end{cases}$$

It follows directly that W encodes the weight function w, concluding the proof. □

Finally, we show that Sum and Average for Reg$_{\mathbb{T}}$ weight functions are in $\mathrm{FP}^{\#\mathrm{P}}$.

Theorem 7.8. *Sum*[VSA, *Reg*] *and Average*[VSA, *Reg*] *are in* $\mathrm{FP}^{\#\mathrm{P}}$.

Proof. We will begin by showing that Sum[VSA, Reg] is in $\mathrm{FP}^{\#\mathrm{P}}$ if all weights assigned by w are natural numbers. We will use this as an oracle for the general upper bound. Let A be a VSet-automaton, $d \in \Sigma^*$ be a document and $w \in \mathrm{Reg}$ be a weight function, which only assigns natural numbers and is represented by a weighted VSet-automaton W. A counting Turing Machine M for solving the problem in #P would have $w(d, \mathrm{t})$ accepting runs for every tuple in $A(d)$. More precisely, M guesses a d-tuple t over $\mathsf{Vars}(A)$ and can check whether $\mathrm{t} \in \llbracket A \rrbracket(d)$ and $w(d, \mathrm{t}) > 0$. If so, M branches into $w(d, \mathrm{t})$ accepting branches. Otherwise, M rejects. Per construction, M has exactly $w(d, \mathrm{t})$ accepting branches for every tuple $\mathrm{t} \in \llbracket A \rrbracket(d)$ with $w(d, \mathrm{t}) > 0$. Thus, the number of accepting runs is exactly $\sum_{\mathrm{t} \in \llbracket A \rrbracket(d)} w(d, \mathrm{t}) = \mathrm{Sum}(\llbracket A \rrbracket, d, w)$.

Now, let $w \in \mathrm{Reg}$ be a weight function, represented by the weighted VSet-automaton W. We can assume, w.l.o.g., that all rationals in W have the denominator d_{lcm}.[16] We recall that $w(d, \mathrm{t}) = \llbracket W \rrbracket(d, \pi_{\mathsf{Vars}(W)}(\mathrm{t}))$. Thus, $w(d, \mathrm{t})$ is the product of $|d| + 1 + 2 * |\mathsf{Vars}(A)|$ rationals, where each factor has the denominator d_{lcm}. Therefore, $\llbracket W \rrbracket(d, \pi_{\mathsf{Vars}(W)}(\mathrm{t}))$ must have the denominator $d_{\mathrm{lcm}}^{|d|+1+2|\mathsf{Vars}(A)|}$[17], which has an encoding length linear in W and d. Thus, Sum[VSA, Reg] can be computed by two calls to a Sum[VSA, Reg]-oracle. The first call only considers positive numerators, whereas the second call only considers negative numerators. Then, Sum[VSA, Reg] is the difference of the results of both oracle calls, divided by $d_{\mathrm{lcm}}^{|d|+1+2*|\mathsf{Vars}(A)|}$.

The upper bound for Average[VSA, Reg$_{\mathbb{T}}$] is immediate from the upper bound of Sum[VSA, Reg$_{\mathbb{T}}$] and Theorem 3.1. □

7.4. **Quantile Aggregation.** The situation for q-Quantile is different from the other aggregation problems, since it remains hard, even when both the spanner and weight function are unambiguous. The reason is that the problem reduces to counting the number of paths in a weighted DAG that are shorter than a given target weight, which is #P-complete due to Mihalák et al. [MSW16].

[16] This can be achieved by computing the least common multiple of all denominators d_{lcm} in W and expanding all fractions $\frac{a}{b}$ by $\frac{b}{d_{\mathrm{lcm}}}$.

[17] For the tropical semiring the denominator is actually d_{lcm}, as the multiplicative operation is $+$, which does not increase the denominator if both summands have the same denominator.

Theorem 7.9. *q-Quantile[uVSA, UReg] is #P-hard under Turing reductions, for every $0 < q < 1$.*

At the core of the quantile problem is the problem of counting up to a threshold $k \neq \infty$:

$$\mathrm{Count}_{<k}(S, d, w) := |\{\mathrm{t} \in P(d) \mid w(d, \mathrm{t}) \leq k\}|.$$

The problems $\mathrm{Count}_{>k}(S, d, w)$ and $\mathrm{Count}_{=k}(S, d, w)$ are defined analogously. The decision problem $\textsc{Count}_{<k}[\mathcal{S}, \mathcal{W}]$ is defined analogously to $\textsc{Sum}[\mathcal{S}, \mathcal{W}]$. We begin by showing that $\textsc{Count}_{<k}[\mathrm{uVSA}, \mathrm{UReg}_{\mathbb{Q}}]$ and $\textsc{Count}_{<k}[\mathrm{uVSA}, \mathrm{UReg}_{\mathbb{T}}]$ #P-hard under Turing reductions. To this end, we reduce from #Partition and #-Product-Partition.

Given a set $N = \{n_1, \ldots, n_n\}$ of natural numbers. Two sets N_1, N_2 are a partition of N if $N_1 \cup N_2 = N$ and $N_1 \cap N_2 = \emptyset$. Furthermore, a partition is perfect, if the sums of the natural numbers in both sets are equal. Given such a set $N = \{n_1, \ldots, n_n\}$, the #Partition problem asks for the number of perfect partitions.

Analogously, a partition N_1, N_2 is called a perfect product partition, if the products of the natural numbers in both sets are equal. Furthermore, the Product-Partition Problem asks whether there is a perfect product partition and the problem #Product-Partition asks for the number of perfect product partitions.

Proposition 7.10. *#Partition and #Product-Partition are #P-complete under Turing reductions.*

Proof. Mihalák et al. [MSW16, Theorem 1] show that #Partition is #P-complete.

The #P-completeness of #Product-Partition follows by a reduction of Ng et al. [NBCK10, Theorem 1], who give a reduction from Exact Cover by 3-sets (X3C) to Product-Partition. We note that this reduction is weakly parsimonious, as defined by Hunt et al. [HMRS98, Definition 2.5]. That is, for every solution of an X3C instance, there are exactly 2 solutions for the constructed Product-Partition instance. Furthermore, Hunt et al. [HMRS98, implicit in Theorem 3.8] show that #X3C is #P-hard. Therefore, the reduction of Ng et al. [NBCK10, Theorem 1] can be used to give a Turing reduction from #X3C to #Product-Partition, which implies that #Product-Partition is also #P-hard under Turing reductions. It is easy to see that #Product-Partition is in #P. □

Lemma 7.11. *Let $k \in \mathbb{Q}$. Then $\textsc{Count}_{<k}[\mathrm{uVSA}, \mathrm{UReg}_{\mathbb{T}}]$ is #P-hard under Turing reductions.*

Proof. We use the same idea as Mihalák et al. [MSW16, Theorem 1] to encode #Partition. Let $N = \{n_1, \ldots, n_n\}$ be an instance of #Partition. Let $d = a^n$. We construct A and W such that every tuple $\mathrm{t} \in [\![A]\!](d)$ corresponds to a partition of N. Furthermore, $w(d, \mathrm{t}) = k$ if and only if the partition encoded by t is perfect.

More formally, $A := (\Sigma, V, Q, q_0, Q_F, \delta)$, where $\Sigma := \{a\}, V := \{x_1, \ldots, x_n\}, Q := \{q_i^j \mid 1 \leq i \leq n, 1 \leq j \leq 5\}$, where $q_i^5 = q_{i+1}^1$ for all $1 \leq i < n$, $q_0 := q_1^1, Q_F := \{q_n^5\}$, and for $1 \leq i \leq n$, δ is defined as follows:

$$\delta(q_i^j, \sigma) := \begin{cases} \{q_i^2\} & \text{if } 1 \leq i \leq n, \sigma = \triangleright_{x_i}, \text{ and } j = 1 \\ \{q_i^3\} & \text{if } 1 \leq i \leq n, \sigma = \triangleleft_{x_i}, \text{ and } j = 2 \\ \{q_i^4\} & \text{if } 1 \leq i \leq n, \sigma = a, \text{ and } j = 2 \\ \{q_i^5\} & \text{if } 1 \leq i \leq n, \sigma = a, \text{ and } j = 3 \\ \{q_i^5\} & \text{if } 1 \leq i \leq n, \sigma = \triangleleft_{x_i}, \text{ and } j = 4\,. \end{cases}$$

Recall, that $q_i^5 = q_{i+1}^1$ for all $1 \leq i < n$.

Furthermore, we define the weighted VSet-automaton W encoding w the same way as A. That is, all transitions labeled by a variable operation $x \in \Gamma_V$ are assigned weight $\overline{1}$, $\delta(q_i^3, a, q_i^5) = n_i$ and $\delta(q_i^2, a, q_i^4) = -n_i$, the initial- and final weight functions:

$$I(q) := \begin{cases} \overline{1} & \text{if } q = q_0 \\ \overline{0} & \text{otherwise ;} \end{cases}$$

$$F(q) := \begin{cases} k & \text{if } q \in Q_F \\ \overline{0} & \text{otherwise .} \end{cases}$$

We observe that every tuple $\mathrm{t} \in [\![A]\!](d)$ encodes a partition of N, that is, $n_i \in N_1$ if $\mathrm{t}(x_i) = [i, i\rangle$ and $n_i \in N_2$ if $\mathrm{t}(x_i) = [i, i+1\rangle$. Furthermore, for every tuple $\mathrm{t} \in [\![A]\!](d)$, the weight $w(d, \mathrm{t})$ is exactly k plus the difference of the sum of all elements in N_1 and the sum of all elements in N_2. We make some observations about A, d, and w.

(1) The number of perfect partitions is exactly $\mathrm{Count}_{=k}([\![A]\!], d, w)$;
(2) $\mathrm{Count}_{<k}([\![A]\!], d, w) = \mathrm{Count}_{>k}([\![A]\!], d, w)$;
(3) $\mathrm{Count}([\![A]\!], d) = 2 \cdot \mathrm{Count}_{<k}([\![A]\!], d, w) + \mathrm{Count}_{=k}([\![A]\!], d, w)$;
(4) $\mathrm{Count}([\![A]\!], d) = 2^{n+1}$;
(5) $\mathrm{Count}_{=k}([\![A]\!], d, w) = 2^{n+1} - 2 \cdot \mathrm{Count}_{<k}([\![A]\!], d, w)$.

Due to Observations (1) and (5) it follows that the number of perfect partitions can be computed by a single call to a $\mathrm{Count}_{<k}([\![A]\!], d, w)$-oracle.

It remains to argue that the observations $(1) - (5)$ hold. Observation (1) follows directly from the previous observation that the weight of each tuple is k plus the difference of the sum of all elements in N_1 and the sum of all elements in N_2. Observation (2) follows from the fact that the partition problem is symmetric, that is for every partition N_1, N_2 of N there is also a partition N_2, N_1. Observation (3) follows from (2), and (4) from the fact that there are 2^n subsets of N and therefore $2 \cdot 2^n$ possible partitions. The last observation (5) follows from (3) and (4). This concludes the proof. □

Along the same lines we show that $\textsc{Count}_{<1}[\mathrm{uVSA}, \mathrm{UREG}_{\mathbb{Q}}]$ is #P-hard under Turing reductions. Note that we do not show hardness for $\textsc{Count}_{<k}[\mathrm{uVSA}, \mathrm{UREG}_{\mathbb{Q}}]$, but only for the case $k = 1$.[18]

Lemma 7.12. *$\textsc{Count}_{<1}[\mathrm{uVSA}, \mathit{UREG}_{\mathbb{Q}}]$ is #P-hard under Turing reductions.*

Proof. Let N be an instance of #Product-Partition. We construct A, d, w and W, as constructed in the proof of Lemma 7.11. However in W, $\delta(q_i^3, a, q_i^5) = n_i$ and $\delta(q_i^2, a, q_i^4) = \frac{1}{n_i}$. Observe, that $w(d, \mathrm{t})$ is exactly the product of all elements in N_1 divided by the product of all elements in N_2, where $n_i \in N_1$ if and only if $\mathrm{t}(x_i) = [i, i\rangle$ and $n_i \in N_2$ if and only if $\mathrm{t}(x_i) = [i, i+1\rangle$. Therefore, the number of perfect product partitions is exactly the number of tuples $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = 1$. Using the same argument as in the proof of Lemma 7.11, it follows that

$$\#\text{Product-Partition} = 2^{n+1} - 2 \cdot \mathrm{Count}_{<1}([\![A]\!], d, w) \ ,$$

and thus, #Product-Partition can be computed by a single $\textsc{Count}_{<1}[\mathrm{uVSA}, \mathrm{UREG}_{\mathbb{Q}}]$-oracle call. □

[18] Recall that, in the proof for the tropical semiring, we add k to all accepting runs by having $F(q) = k$, if $q \in Q_F$. This is not possible over the numerical semiring, as the multiplicative operation is the numerical multiplication $\cdot$ and not the numerical addition $+$.

The following corollary follows directly from Lemmas 7.11 and 7.12.

Corollary 7.13. *$\textsc{Count}_{<1}$[uVSA, UREG] is #P-hard under Turing reductions.*

We are finally ready to give the proof of Theorem 7.9.

Proof of Theorem 7.9. We show that $\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w)$ can be computed in polynomial time, using a q-QUANTILE[uVSA, UREG]-oracle therefore, concluding that the problem q-QUANTILE[uVSA, UREG] is also #P-hard under Turing reductions.

Let $A \in$ uVSA, $d \in \Sigma^*$, and $w \in$ UREG represented by an unambiguous weighted VSet-automaton W. Furthermore, let $0 < q < 1$, such that $q = \frac{a}{b}$. Due to Theorem 3.1, $c := \mathrm{Count}(\llbracket A \rrbracket, d)$ can be computed in polynomial time. Let $0 \leq r \leq c \cdot (b-1)$. By Lemma 4.7[19], there are VSet-automata $A_r, A'_r \in$ uVSA and documents d_r, d'_r, such that $\mathrm{Count}(\llbracket A_r \rrbracket, d_r) = r$ and $\mathrm{Count}(\llbracket A'_r \rrbracket, d'_r) = c \cdot (b-1) - r$. Let W_r (resp., W'_r) be A_r (resp., A'_r), interpreted as unambiguous weighted VSet-automaton, where all transitions of A_r (resp., A'_R) have weight $\overline{1}$, the initial weight function assigns weight $\overline{1}$ to the initial state of A_r (resp., A'_r), and the final weight function assigns weight 0 (resp., 1)[20] to all accepting states of A_r (resp., A'_r). Slightly overloading notation, we define

$$A' := (A \cdot d_r \cdot d'_r) \vee (d \cdot A_r \cdot d'_r) \vee (d \cdot d_r \cdot A'_r)$$

and

$$W' := (W \cdot d_r \cdot d'_r) \vee (d \cdot W_r \cdot d'_r) \vee (d \cdot d_r \cdot W'_r)$$

It is straightforward to verify that both, A' and W' are unambiguous. Let $d' = d \cdot d_r \cdot d'_r$ and let w' (resp, w_r, w'_r) be the weight function, represented by W' (resp., W_r, W'_r). It follows from the definition that

$$\begin{aligned}\mathrm{Count}(\llbracket A' \rrbracket, d') &= \mathrm{Count}(\llbracket A \rrbracket, d) + \mathrm{Count}(\llbracket A_r \rrbracket, d_r) + \mathrm{Count}(\llbracket A'_r \rrbracket, d'_r) \\ &= c + r + (c \cdot (b-1) - r) = c \cdot b \ .\end{aligned}$$

Furthermore, recalling that $w(d, \mathrm{t}) = 0$ for all tuples $\mathrm{t} \in \llbracket A_r \rrbracket(d_r)$ and $w(d, \mathrm{t}) = 1$ for all tuples $\mathrm{t} \in \llbracket A'_r \rrbracket(d'_r)$, we have that

$$\begin{aligned}&\mathrm{Count}_{<1}(\llbracket A' \rrbracket, d', w') \\ &= \mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + \mathrm{Count}_{<1}(\llbracket A_r \rrbracket, d_r, w_r) + \mathrm{Count}_{<1}(\llbracket A'_r \rrbracket, d'_r, w'_r) \\ &= \mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + r + 0 \ .\end{aligned}$$

Using binary search, we compute r_{min} as the smallest r with q-Quantile$(\llbracket A' \rrbracket, d', w') < 1$. Thus,

$$\frac{\mathrm{Count}_{<1}(\llbracket A' \rrbracket, d', w')}{\mathrm{Count}(\llbracket A' \rrbracket, d')} = \frac{\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + r_{min}}{c \cdot b} \geq q \ .$$

For the sake of contradiction, assume that $\frac{\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + r_{min}}{c \cdot b} > q = \frac{c \cdot a}{c \cdot b}$. It follows that, $\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + r_{min} > c \cdot a$ and therefore, as all involved numbers are natural numbers, $\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + r_{min} - 1 \geq c \cdot a$. Thus, $\frac{\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + (r_{min} - 1)}{c \cdot b} \geq q$, leading to the desired contradiction, as r_{min} was assumed to be minimal.

We have that $\frac{\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) + r_{min}}{c \cdot b} = q = \frac{c \cdot a}{c \cdot b}$. It follows that

$$\mathrm{Count}_{<1}(\llbracket A \rrbracket, d, w) = c \cdot a - r_{min} \ ,$$

[19] For instance with $v = \mathsf{Vars}(A) \cdot b$.

[20] Note that we use 0 and 1 instead of $\overline{0}$ and $\overline{1}$ on purpose. The reason is that we want to assign the same weights for both semirings.

which concludes the proof. □

8. Aggregate Approximation

Now that we have a detailed understanding on the complexity of computing exact aggregates, we want to see in which cases the result can be approximated. We only consider the situation where the exact problems are intractable and want to understand when the considered aggregation problems can be approximated by fully polynomial-time randomized approximation schemes (FPRAS), and when the existence of such an FPRAS would contradict commonly believed conjectures, like RP $\neq$ NP and the conjecture that the polynomial hierarchy does not collapse.

Based on the results for the computation of exact aggregates, we can already give some insights into the possibility of approximation. That is, Zuckerman [Zuc96] shows that #SAT can not be approximated by an FPRAS unless NP = RP. Furthermore, as shown by Dyer et al. [DGGJ04], this characterization extends to all problems which are #P-complete under parsimonious reductions. Therefore, due to Theorems 5.4, and 7.7, we have the following corollary.

Corollary 8.1. *Unless* NP = RP*, the problems* $\textsc{Sum}[\text{VSA}, \textsc{CWidth}]$*,* $\textsc{Sum}[\text{uVSA}, \textsc{Reg}_{\mathbb{T}}]$*, and* $\textsc{Average}[\text{uVSA}, \textsc{Reg}_{\mathbb{T}}]$ *do not have an* FPRAS*.*

Arenas et at. [ACJR19, Corollary 3.3] showed that every function in spanL admits an FPRAS. Therefore, due to Theorem 5.5, we have the following corollary.

Corollary 8.2. $\textsc{Sum}[\text{VSA}, \textsc{CWidth}_{\mathbb{N}}]$ *has an* FPRAS*.*

In the remainder of this section, we will revisit the other intractable cases of spanner aggregation and study whether or not approximation is possible.

8.1. Approximation is Hard at First Sight.

We begin with some inapproximability results. For instance, as we show now, the existence of an FPRAS for the problems Min, Max with $\textsc{Reg}_{\mathbb{Q}}$ weight functions would imply a collapse of the polynomial hierarchy, even when spanners are unambiguous. Furthermore, for Max and $\textsc{Reg}_{\mathbb{T}}$ weight functions the same result holds.

Theorem 8.3. $\textsc{Min}[\text{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$ *and* $\textsc{Max}[\text{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$ *do not have an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. Assume there is an FPRAS for $\textsc{Min}[\text{uVSA}, \textsc{Reg}_{\mathbb{Q}}]$. We will show that such an FPRAS implies that the NP-complete problem SAT is in BPP, which implies that the polynomial hierarchy collapses to the second level.[21]

Let $\phi(x_1, \ldots, x_n)$ be a Boolean formula, given in CNF, and let A, d, and W' be as defined in the proof for $\textsc{Max}[\text{uVSA}, \textsc{Reg}_{\mathbb{T}}]$ of Theorem 7.3, where W' is interpreted as a weighted VSet-automaton over the numerical semiring. Observe that, due to $\overline{1} = 1$ and $\overline{0} = 0$, it follows that $[\![W']\!]_{\mathbb{Q}}(d, \mathrm{t}) \geq 1$ if the valuation α_{t} encoded by t does not satisfy at least one clause of ϕ and 0 otherwise. Let w be the weight function encoded by W'.

[21] NP $\subseteq$ BPP implies that PH $\subseteq$ BPP (cf. Zachos [Zac88]) and as BPP $\subseteq (\Pi_2^{\mathrm{P}} \cap \Sigma_2^{\mathrm{P}})$ (cf. Lautemann [Lau83]) the polynomial hierarchy collapses on the second level. Furthermore, as BPP is closed under complement, coNP $\subseteq$ BPP implies that NP $\subseteq$ BPP resulting in the same collapse of the polynomial hierarchy.

For the sake of contradiction, assume that there is an FPRAS for $\textsc{Min}[\text{uVSA}, \text{REG}_{\mathbb{Q}}]$ and let $\delta = 0.4$. Assume that ϕ is satisfiable, thus $\text{Min}([\![A]\!], d, w) = 0$. Then the FPRAS must return 0 with probability at least $\frac{3}{4}$. On the other hand, if ϕ is not satisfiable, the FPRAS must return a value $x \geq (1-\delta) \cdot 1 = 0.6$ with probability at least $\frac{3}{4}$. Consider the algorithm which calls the FPRAS and accepts if the approximation is 0, and rejects otherwise. This algorithm is a BPP algorithm for SAT, resulting in the desired contradiction.

The proof for $\textsc{Max}[\text{uVSA}, \text{REG}_{\mathbb{Q}}]$ is analogous. The only difference is that the final weight function of W' is multiplied by -1, that is, W' assigns weight $-x$ to each tuple, encoding a valuation α which does not satisfy x clauses of ϕ. □

Theorem 8.4. *$\textsc{Max}[\text{uVSA}, \text{REG}_{\mathbb{T}}]$ cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. Let $\phi(x_1, \ldots, x_n)$ be a Boolean formula, given in CNF. We assume, w.l.o.g., that the valuation which assigns false to all variables does not satisfy ϕ. Let A, d, and w be as defined in the proof for $\textsc{Max}[\text{uVSA}, \text{REG}_{\mathbb{T}}]$ in the proof of Theorem 7.3. Thus, $\text{Max}([\![A]\!], d, w) \geq 1$ if ϕ is satisfiable and $\text{Max}([\![A]\!], d, w) = 0$ if ϕ is not satisfiable.

For the sake of contradiction, assume that there is an FPRAS for $\textsc{Max}[\text{uVSA}, \text{REG}_{\mathbb{T}}]$ and let $\delta = 0.4$. Assume that ϕ is satisfiable, thus $\text{Max}([\![A]\!], d, w) \geq 1$. Then the FPRAS must return a value $x \geq (1-\delta) \cdot 1 = 0.6$ with probability at least $\frac{3}{4}$. On the other hand, if ϕ is not satisfiable, the FPRAS must return 0 with probability at least $\frac{3}{4}$. Therefore, we can obtain a BPP algorithm for SAT as follows. The algorithm first calls the FPRAS, accepts if the approximation is bigger than 0, and rejects otherwise. □

Concerning Sum and Average the only case which is not resolved by Corollary 8.1 is the case of $\textsc{Average}[\text{VSA}, \text{CWIDTH}]$. We show now that, under reasonable complexity assumptions, this problem can also not be approximated by an FPRAS.

Theorem 8.5. *$\textsc{Average}[\text{VSA}, \textsc{CWidth}]$ cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. We will show that such an FPRAS implies that the NP-complete problem SAT is in BPP, which implies that the polynomial hierarchy collapses to the second level.

To this end, let A, d and w be as constructed in the proof of Theorem 5.4. Recall that given a propositional formula ϕ in CNF, we have that $\text{Sum}([\![A]\!], d, w) = c$, where c is the number of satisfying assignments of ϕ.

Assume there is an FPRAS for $\textsc{Average}[\text{VSA}, \text{CWIDTH}]$ and let $\delta = 0.5$. Assume that ϕ is not satisfiable. Then the FPRAS on input A, d, w must return 0 with probability at least $\frac{3}{4}$. On the other hand, if ϕ is satisfiable, thus $c > 0$, the FPRAS must return a value $x \geq (1-\delta) * \text{Avg}([\![A]\!], d, w) = \frac{1}{2} \cdot \frac{c}{\text{Count}([\![A]\!], d)} > 0$, with probability at least $\frac{3}{4}$. Therefore, the algorithm which first approximates $\text{Avg}([\![A]\!], d, w)$ with $\delta = 0.5$, rejects if the approximation is 0 and accepts otherwise is a BPP algorithm for SAT, implying that NP $\subseteq$ BPP, which implies that the polynomial hierarchy collapses to the second level. □

We now turn to the quantile problem. It turns out that this problem is difficult to approximate even if the weight functions only return 0 or 1.

Theorem 8.6. *Let $0 < q < 1$. Then, q-$\textsc{Quantile}[\text{VSA}, \textsc{CWidth}]$ cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses to the second level.*

Proof. We will show that an FPRAS for q-QUANTILE[VSA, CWIDTH] implies a BPP algorithm for SAT. To this end, let ϕ be a propositional formula ϕ in CNF. Assume that $q = \frac{1}{2}$ and let A and d be as constructed in the proof of Theorem 5.4. However, let w be the weight function which is represented by the $\mathbb{Q}$-Relation R over $\{x\}$ with

$$R(d) := \begin{cases} 1 & \text{if } d = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Recall from the construction of A and d that A is the union of two automata A_1, A_{-1}, such that $\text{Count}(\llbracket A_1 \rrbracket, d) = 2^n$ and $\text{Count}(\llbracket A_{-1} \rrbracket, d) = s$, where s is the number of non-satisfying assignments for ϕ, furthermore, $\mathrm{t} \in \llbracket A_1 \rrbracket(d)$ if and only if $d_{\mathrm{t}(x)} = 1$ and $\mathrm{t} \in \llbracket A_{-1} \rrbracket(d)$ if and only if $d_{\mathrm{t}(x)} = -1$. We observe that $R(-1) = 0$ and therefore, for every $\mathrm{t} \in \llbracket A \rrbracket(d)$ we have that

$$w(d, \mathrm{t}) = \begin{cases} 1 & \text{if } \mathrm{t} \in \llbracket A_1 \rrbracket(d) \\ 0 & \text{if } \mathrm{t} \in \llbracket A_{-1} \rrbracket(d) \ . \end{cases}$$

Thus, $\frac{1}{2}$-Quantile$(\llbracket A \rrbracket, d, w) = 0$ if and only if ϕ is not satisfiable.

Assuming there is an FPRAS for q-QUANTILE[VSA, CWIDTH], one can decide SAT with a probability of $\frac{3}{4}$ by approximating q-Quantile$(\llbracket A \rrbracket, d, w)$ with $\delta = 0.5$, rejecting if the approximation is 0 and accepting otherwise. This, however, implies that NP $\subseteq$ BPP, which implies a collapse of the polynomial hierarchy on the second level.

The general case for $0 < q < 1$ follows by slightly adopting the previous construction. That is, assume that $q = \frac{a}{b}$. Due to $0 \leq q \leq 1$, it must hold that $1 \leq a < b$. We construct a VSet-automaton A' and a document d' as follows. Let $\sigma \notin \Sigma$ be a new alphabet symbol. The document d' consists of b copies of d, separated by σ and A' consists of a copies of A_{-1} and $b - a$ copies of A_1. More formally,

$$d' := (d \cdot \sigma)^b \ .$$

Furthermore, slightly abusing notation, we define

$$A' := (A_{-1} \cdot \sigma)^a \cdot (A_1 \cdot \sigma)^{b-a} \ .$$

We observe that on input document d', the automaton A' accepts exactly $2^n \cdot (b - a)$ tuples t with $w(d', \mathrm{t}) = 1$ and $s \cdot a$ tuples with weight 0. Therefore, $\frac{a}{b}$-Quantile$(S, d, w) = 0$ if and only if

$$\frac{s \cdot a}{2^n \cdot (b - a) + s \cdot a} \geq \frac{a}{b} \ .$$

Solving this equation for s, it holds that $\frac{a}{b}$-Quantile$(S, d, w) = 0$ if and only if $s = 2^n$ and therefore $\frac{a}{b}$-Quantile$(S, d, w) = 0$ if and only if ϕ is not satisfiable.

The rest of the proof is analogous to the case that $q = \frac{1}{2}$. □

When the spanners are unambiguous, the simplest intractable case for q-QUANTILE is the one with UREG weight functions (see Table 1). Again, we can show that approximation is hard.

Theorem 8.7. *Let $0 < q < 1$. Then, q-QUANTILE[uVSA, UREG$_{\mathbb{T}}$] cannot be approximated by an* FPRAS*, unless the polynomial hierarchy collapses on the second level.*

Proof. We show that an FPRAS for q-QUANTILE[uVSA, UREG$_{\mathbb{T}}$] implies a BPP algorithm for the NP-complete Partition problem. To this end, let $S = \{s_1, \ldots, s_n\}$ be a set of natural

numbers. Furthermore, let A, d, w be constructed from S as in the proof of Lemma 7.11 with $k = 0$.

Per construction of A, d and w, every tuple $\mathrm{t} \in [\![A]\!](d)$ corresponds to a partition of S, such that the partition is perfect if and only if $w(d, \mathrm{t}) = 0$. Furthermore, due to the partition problem being symmetrical, for every tuple $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = k$ there is a tuple $\mathrm{t}' \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = -k$. Thus, $\frac{1}{2}$-Quantile$([\![A]\!], d, w) = 1$ if and only if there is a tuple $\mathrm{t} \in [\![A]\!](d)$ with $w(d, \mathrm{t}) = 0$.

Let $q = \frac{1}{2}$. Assuming there is an FPRAS for q-Quantile[uVSA, UReg$_{\mathbb{T}}$], one can decide Partition with a probability of $\frac{3}{4}$ by approximating q-Quantile$([\![A]\!], d, w)$ with $\delta = 0.5$, accepting if the approximation is 0 rejecting otherwise. This implies that the algorithm accepts if and only if there is a perfect partition and therefore, NP $\subseteq$ BPP, which implies a collapse of the polynomial hierarchy on the second level.

For the general case, assume that $q = \frac{a}{b}$. We observe that due to $0 < q < 1$, it must hold that $a < b$. By Observation (4) in the proof of Lemma 7.11, Count$([\![A]\!], d) = 2^{n+1}$. As in the proof of Theorem 7.9, we construct a VSet-automaton A', a document d' and a weight function w', represented by the weighted automaton $W' \in \mathrm{UReg}_{\mathbb{T}}$, such that q-Quantile$(A', d', w') = 0$ if and only if S has a perfect partition. By Lemma 4.7, there are VSet-automata $A_{-1}, A_1 \in$ uVSA and documents $d_{-1}, d_1 \in \Sigma^*$ such that Count$([\![A_{-1}]\!], d_{-1}) = (a-1) \cdot 2^n$ and Count$([\![A_1]\!], d_1) = (b-a-1) \cdot 2^n$. Let W_{-1} (resp., W_1) be the same as A_{-1} (resp., A_1) interpreted as weighted automaton over the tropical semiring, such that all transitions are assigned weight 0 and the final weight function assigns weight -1 (resp., 1) to all accepting states. Let w_{-1} (resp., w_1) be the weight function, represented by W_{-1} (resp., W_1) Thus, $w_{-1}(d_{-1}, \mathrm{t}) = -1$ if and only if $\mathrm{t} \in [\![A_{-1}]\!](d_{-1})$ and $w_1(d_1, \mathrm{t}) = 1$ if and only if $\mathrm{t} \in [\![A_1]\!](d_1)$. Let σ be a new alphabet symbol. We construct A', d', and W' as follows.

$$
\begin{aligned}
d' &= d_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot d_1 \\
A' &= (A_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot A \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot A_1) \\
W' &= (W_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot W \cdot \sigma \cdot d_1) \vee (d_{-1} \cdot \sigma \cdot d \cdot \sigma \cdot W_1) \,.
\end{aligned}
$$

Furthermore, let w' be the weight function, represented by W'. It follows that

$$
\begin{aligned}
\mathrm{Count}_{<0}([\![A']\!], d', w') &= (a-1) \cdot 2^n + \mathrm{Count}_{<0}([\![A]\!], d, w) \\
\mathrm{Count}_{\leq 0}([\![A']\!], d', w') &= (a-1) \cdot 2^n + \mathrm{Count}_{\leq 0}([\![A]\!], d, w) \\
\mathrm{Count}([\![A']\!], d') &= (a-1) \cdot 2^n + 2 \cdot 2^n + (b-a-1) \cdot 2^n = b \cdot 2^n \,.
\end{aligned}
$$

We make a case distinction on S. If S has a perfect partition, $\mathrm{Count}_{<0}([\![A]\!], d, w) < 2^n$ and $\mathrm{Count}_{\leq 0}([\![A]\!], d, w) \geq 2^n$. Thus, q-Quantile$(A', d', w') = 0$. Otherwise, if S has no perfect partition, $\mathrm{Count}_{<0}([\![A]\!], d, w) = 2^n$ and therefore q-Quantile$(A', d', w') < 0$. Therefore, q-Quantile$(A', d', w') = 0$ if and only if S has a perfect partition. This concludes the proof. □

We note that the case of approximating q-Quantile[uVSA, UReg$_{\mathbb{Q}}$] does not follow analogous to the proof for q-Quantile[uVSA, UReg$_{\mathbb{T}}$]. The main reason is the fact that #Partition can be encoded into a weight function automaton $w_{\mathbb{T}} \in \mathrm{UReg}_{\mathbb{T}}$, such that perfect partitions correspond to tuples with weight 0, whereas #Product-Partition is encoded into a weight function $w_{\mathbb{Q}} \in \mathrm{UReg}_{\mathbb{Q}}$, such that perfect product partitions correspond to tuples with weight 1. Furthermore, all weights assigned by $w_{\mathbb{T}}$ are integers, whereas $w_{\mathbb{Q}}$ assigns

rational numbers. Therefore it is not obvious whether or not q-Quantile[uVSA, UReg$_{\mathbb{Q}}$] can be approximated by an FPRAS. This case is left open for future research.

8.2. **When an FPRAS is Possible.** We show that Theorem 8.5 is very much on the intractability frontier: it shows that approximation is intractable if weight functions can assign 1 and -1. On the other hand, if the weight functions are restricted to *nonnegative* numbers, then approximating Sum and Average is possible with an FPRAS.

Theorem 8.8. *Sum*[VSA, *CWidth*$_{\mathbb{Q}_+}$] *and Average*[VSA, *CWidth*$_{\mathbb{Q}_+}$] *can be approximated by an* FPRAS.

Proof. From Corollary 8.2 and Theorem 3.1 we conclude that there is an FPRAS for each of the problems Sum[VSA, CWidth$_{\mathbb{N}}$] and Count[VSA]. We will use these FPRAS to give an FPRAS for Sum[VSA, CWidth$_{\mathbb{Q}^+}$] and Average[VSA, CWidth$_{\mathbb{Q}^+}$].

In the following, we will denote an FPRAS approximation with error rate δ of the problem Count($[\![A]\!], d$) (resp., Sum($[\![A]\!], d, w$) and Avg($[\![A]\!], d, w$)) by Count($[\![A]\!], d, \delta$) (resp., Sum($[\![A]\!], d, w, \delta$) and Avg($[\![A]\!], d, w, \delta$)).

We begin by showing that Sum[VSA, CWidth$_{\mathbb{Q}^+}$] admits an FPRAS. Let $A \in$ VSA be a VSet-automaton, $d \in \Sigma^*$ be a document, and $w \in$ CWidth$_{\mathbb{Q}^+}$ be a weight function. Recall that every weight $x \in \mathbb{Q}^+$ is encoded by its numerator and its denominator. Let D be the set of denominators used by w and let lcm be the least common multiple of all elements in D. We note that, as argued in the proof of Theorem 7.8, lcm can be computed in polynomial time. Let $w_{\mathbb{N}}(d, \mathrm{t}) = w(d, \mathrm{t}) \cdot \mathrm{lcm}$. Per definition of lcm, $w_{\mathbb{N}} \in$ CWidth$_{\mathbb{N}}$ only assigns natural numbers. Furthermore, $w(d, \mathrm{t}) = \frac{w_{\mathbb{N}}(d,\mathrm{t})}{\mathrm{lcm}}$. It follows that $\mathrm{Sum}([\![A]\!], d, w, \delta) := \frac{\mathrm{Sum}([\![A]\!],d,w_{\mathbb{N}},\delta)}{\mathrm{lcm}}$ is an δ-approximation of $\mathrm{Sum}(S, d, w)$ with success probability $\frac{3}{4}$, concluding this part of the proof.

It remains to show that Average[VSA, CWidth$_{\mathbb{Q}_+}$] admits an FPRAS. We show that the algorithm which, with success rate $(\frac{3}{4})^{0.5}$, calculates a $\frac{\delta}{3}$-approximations for Count and Sum, and then returns the quotient of the results, is an FPRAS for the problem Average[VSA, CWidth$_{\mathbb{Q}_+}$]. We note that the probability that both approximations are successful is $(\frac{3}{4})^{0.5} \cdot (\frac{3}{4})^{0.5} = \frac{3}{4}$.

It remains to show that the quotient of both results, $\mathrm{Avg}([\![A]\!], d, w, \delta) := \frac{\mathrm{Sum}([\![A]\!],d,w,\frac{\delta}{3})}{\mathrm{Count}([\![A]\!],d,\frac{\delta}{3})}$, is indeed a δ-approximation of $\mathrm{Avg}([\![A]\!], d, w)$. Formally, we have to show that

$$(1-\delta) \cdot \mathrm{Avg}(S, d, w) \leq \mathrm{Avg}([\![A]\!], d, w, \delta) \leq (1+\delta) \cdot \mathrm{Avg}([\![A]\!], d, w) \; .$$

We begin with the first inequality:

$$\begin{aligned} \mathrm{Avg}([\![A]\!], d, w, \delta) &= \frac{\mathrm{Sum}([\![A]\!], d, w, \frac{\delta}{3})}{\mathrm{Count}([\![A]\!], d, \frac{\delta}{3})} \geq \frac{(1-\frac{\delta}{3}) \cdot \mathrm{Sum}([\![A]\!], d, w)}{(1+\frac{\delta}{3}) \cdot \mathrm{Count}([\![A]\!], d)} \\ &= \frac{1-\frac{\delta}{3}}{1+\frac{\delta}{3}} \cdot \frac{\mathrm{Sum}([\![A]\!], d, w)}{\mathrm{Count}([\![A]\!], d)} \geq (1-\delta) \cdot \mathrm{Avg}([\![A]\!], d, w) \; . \end{aligned}$$

Algorithm 2: PositionalQuantileApprox(A, d, w, q, δ)

Input: $A \in \text{VSA}, d \in \Sigma^*, w \in \text{Poly}, 0 \leq q \leq 1, 0 \leq \delta \leq 1$
Output: A positional δ-approximation of q-Quantile$([\![A]\!], d, w)$ with success rate $\frac{3}{4}$.

```
W ← {{·}}
for 1 ≤ i ≤ 4 · ⌈ln(16)/(2δ²)⌉ do
    t ← Sample(A, d, δ/3)
    Add w(d, t) to W
if |W| < ⌈ln(16)/(2δ²)⌉ then
    Fail                                   ▷ Sample size too small
Return q-Quantile(W)
```

It is straightforward to verify that $\frac{1-\frac{\delta}{3}}{1+\frac{\delta}{3}} \geq (1-\delta)$ holds for every $0 \leq \delta \leq 1$. The second inequality follows analogously:

$$\begin{aligned}\text{Avg}([\![A]\!], d, w, \delta) &= \frac{\text{Sum}([\![A]\!], d, w, \frac{\delta}{3})}{\text{Count}([\![A]\!], d, \frac{\delta}{3})} \leq \frac{(1+\frac{\delta}{3}) \cdot \text{Sum}([\![A]\!], d, w)}{(1-\frac{\delta}{3}) \cdot \text{Count}([\![A]\!], d)} \\ &= \frac{1+\frac{\delta}{3}}{1-\frac{\delta}{3}} \cdot \frac{\text{Sum}([\![A]\!], d, w)}{\text{Count}([\![A]\!], d)} \leq (1+\delta) \cdot \text{Avg}([\![A]\!], d, w)\ .\end{aligned}$$

Again, it is straightforward to verify that $\frac{1+\frac{\delta}{3}}{1-\frac{\delta}{3}} \leq (1+\delta)$ holds for every $0 \leq \delta \leq 1$. □

Our second positive result is about approximating quantiles *in a positional manner*. Let d be a document, S be a document spanner, w be a weight function and $0 \leq q \leq 1$ with $q \in \mathbb{Q}$. Then, for $\delta > 0$, we say that $k \in \mathbb{Q}$ is a positional δ-approximation of q-Quantile(S, d, w) if there is a $q' \in \mathbb{Q}$, with $q - \delta \leq q' \leq q + \delta$ and $k = q'$-Quantile(S, d, w).[22]

Lemma 8.9 (Hoeffding's Inequality)**.** *Let $X_1, \ldots, X_n$ be independent random variables with $0 \leq X_i \leq 1$ for $1 \leq i \leq n$. Let $X = \Sigma_{i=1}^n X_i$ and let* EX *denote the expectation of X. Then, for any $\lambda > 0$, $Pr(X - \text{EX} \geq \lambda) \leq e^{\frac{-2\lambda^2}{n}}$.*

Theorem 8.10. *Let $0 \leq q \leq 1$. There is a probabilistic algorithm that calculates a positional δ-approximation of q-Quantile[VSA, Poly] with success probability at least $\frac{3}{4}$. Furthermore, the run time of the algorithm is polynomial in the input and $\frac{1}{\delta}$.*

Proof. Let $A \in \text{VSA}$ be a functional VSet-automaton and $d \in \Sigma^*$ be a document. Arenas et al. [ACJR19, Corollary 4.1] showed that given a functional VSet-automaton, one can sample tuples $\text{t} \in [\![A]\!](d)$ uniformly at random with success probability $\geq \frac{1}{2}$.[23] We will use this sampling algorithm to first create a sample of the assigned weights and then return the q-Quantile of this sample. The algorithm is depicted in Algorithm 2.

We note that this algorithm has two points of failure. On one hand, it can happen that less then $s := \lceil \frac{\ln(16)}{2\delta^2} \rceil$ calls to the sampling algorithm of Arenas et al. [ACJR19] are successful.

[22] The idea of positional quantile approximations was originally introduced by Manku et al. [MRL98] in the context of quantile computations with limited memory.

[23] We note that the sampling algorithm by Arenas et al. [ACJR19, Corollary 4.1] detects and reports failures.

On the other hand, it can happen that the returned quantile is no positional δ-approximation of the quantile. We show that both of these points of failure have a probability of less than $\frac{1}{8}$. Thus, the probability that the whole algorithm is successful is $\frac{7}{8} \cdot \frac{7}{8} > \frac{3}{4}$. We will first show that Line 6 is reached with probability less than $\frac{1}{8}$.

The success probability of each call to the sampling algorithm of Arenas et al. [ACJR19] is at least $\frac{1}{2}$. Thus, the expected number of samples, generated by $4s$ consecutive calls to the algorithm is at least $2s$. Using Hoeffding's Inequality, the probability that $4s$ consecutive calls to the sampling algorithm yield less than s samples is less than e^{-s} and therefore less than $\frac{1}{8}$ for every $s \geq 3$.[24]

It remains to show that a total of s samples is enough to guarantee that the q-Quantile of W is a positional δ-approximation of q-Quantile$([\![A]\!], d, w)$ with probability at least $\frac{7}{8}$.

Let $w_{q-\delta} = (q-\delta)$-Quantile$([\![A]\!], d, w)$ and $w_{q+\delta} = (q+\delta)$-Quantile$([\![A]\!], d, w)$. Furthermore, let $W_{q-\delta} = \{\!\{x \in W \mid x < w_{q-\delta}\}\!\}$ and $W_{q+\delta} = \{\!\{x \in W \mid x > w_{q+\delta}\}\!\}$. We say that a sample is bad, if either $|W_{q-\delta}| \geq q \cdot s$ or $|W_{q+\delta}| \geq (1-q) \cdot s$. We will first show that the probability that $|W_{q-\delta}| \geq q \cdot s$ is at most $e^{-2\delta^2 \cdot s}$. For each element $x \in W$ the probability that $x \in W_{q-\delta}$ is at most $(q-\delta)$. Thus, the expected size of $W_{q-\delta}$ is $(q-\delta) \cdot s$. Using Hoeffding's Inequality, with $\lambda = \delta \cdot s$ the probability that $|W_{q-\delta}| \geq q \cdot s$ is at most $e^{-2\delta^2 \cdot s}$. On the other hand, the for each element $x \in W$ the probability that $x \in W_{q+\delta}$ is at most $(1-(q+\delta)) = 1-q-\delta$. Thus, the expected size of $W_{q+\delta}$ is $(1-q-\delta) \cdot s$. Again, using Hoeffding's Inequality, with $\lambda = \delta \cdot s$ the probability that $|W_{q+\delta}| \geq (1-q) \cdot s$ is at most $e^{-2\delta^2 \cdot s}$. Therefore, the probability for a bad sample is at most $2 \cdot e^{-2\delta^2 \cdot s}$. Due to $s = \lceil \frac{\ln(32)}{2\delta^2} \rceil$, the probability of a bad sample is at most $\frac{1}{8}$, concluding the proof. □

9. Conclusions

We investigated the computational complexity of common aggregate functions over regular document spanners given as regex formulas and VSet-automata. While each of the studied aggregate functions is intractable in the general case, there are polynomial-time algorithms under certain general assumptions. These include the assumption that the numerical value of the tuples is determined by a constant number of variables, or that the spanner is represented as an (unambiguous) VSet-automaton. Moreover, we established quite general tractability results when randomized approximations (FPRAS) are possible. The upper bounds that we obtained for general (functional) VSet-automata immediately generalize to aggregate functions over queries that involve relational-algebra operators and string-equality conditions on top of spanners, whenever these inner queries can be *efficiently* compiled into a single VSet-automaton [FKP18, PFKK19]. Moreover, these upper bounds immediately generalize to allow for *grouping* (i.e., the GROUP BY operator) by computing the tuples of the grouping variables and applying the algorithms to each group separately.

We identified several interesting cases where the computation of $\alpha(S(d))$ can avoid the materialization of the exponentially large set $S(d)$, where, d is the document, S is the spanner, and α is the aggregate function. Notably, this is the case (1) for Min with general VSet-spanners and weight functions in $\text{Reg}_{\mathbb{T}}$, UReg, and CWidth, (2) for Max with general VSet-spanners and weight functions in UReg and CWidth, (3) for Sum and

[24] Obviously, we can call the sampling algorithm 16 times for $s = 1$ and $s = 2$ to ensure a failure rate of less than $\frac{1}{8}$.

Average with uVSA-spanners and weight functions in $\text{Reg}_{\mathbb{Q}}$, UReg and CWidth, and (4) for q-Quantile with uVSA-spanners and CWidth weight functions.

Yet, several basic questions are left for future investigation. A natural next step would be to seek additional useful assumptions that cast the aggregate queries tractable: Can monotonicity properties of the numerical functions lead to efficient algorithms in cases that are otherwise intractable? What are the regex formulas that can be efficiently translated into unambiguous VSet-automata (and, hence, allow to leverage the algorithms for such VSet-automata)? Another important direction is to generalize the results in a more abstract framework, such as the *Functional Aggregate Queries* (FAQ) [KNR16], in order to provide a uniform explanation of our findings and encompass general families of aggregate functions rather than specific ones. Finally, the practical side of our work remains to be studied: How do we make our algorithms efficient in practice? How effective is the sampling approach in terms of the balancing between accuracy and execution cost? Can we accurately compute estimators of aggregate functions over (joins of) spanners within the setting of *online aggregation* [HH99, LWYZ16]?

Some of our tractability results reduce the aggregation problems to path problems in DAGs. Since these DAGs are not prohibitively large, we believe that this approach may already be a valid basis for a concrete implementation. Testing empirically whether this is actually the case is, however, a topic for future work.

Acknowledgment

The authors are grateful to Noa Bratman for participating in the initial efforts on the research reported in this manuscript. Furthermore, we thank the anonymous reviewers of ICDT 2021 and LMCS for many helpful remarks. This work was supported by the German-Israeli Foundation for Scientific Research and Development (GIF), grant I-1502-407.6/2019. The work of Johannes Doleschal and Wim Martens was also supported by the Deutsche Forschungsgemeinschaft (DFG), grant 369116833. The work of Benny Kimelfeld was also supported by the Israel Science Foundation (ISF), grants 1295/15 and 768/19, and the DFG project 412400621 (DIP program).

References

[ABMN19] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *22nd International Conference on Database Theory (ICDT)*, pages 22:1–22:19, 2019. doi:10.4230/LIPIcs.ICDT.2019.22.

[ACJR19] Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. Efficient logspace classes for enumeration, counting, and uniform generation. In *Proceedings of the 38th Symposium on Principles of Database Systems (PODS)*, pages 59–73, 2019. doi:10.1145/3294052.3319704.

[CG89] K. Y. Cockwell and I. G. Giles. Software tools for motif and pattern scanning: program descriptions including a universal sequence reading algorithm. *Computer Applications in the Biosciences*, 5(3):227–232, 1989.

[DBKM21] Johannes Doleschal, Noa Bratman, Benny Kimelfeld, and Wim Martens. The Complexity of Aggregates over Extractions by Regular Expressions. In *24th International Conference on Database Theory (ICDT)*, pages 10:1–10:20, 2021. doi:10.4230/LIPIcs.ICDT.2021.10.

[DGGJ04] Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. doi:10.1007/s00453-003-1073-y.

[DKM+19] Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *Proceedings of the 38th Symposium on Principles of Database Systems (PODS)*, pages 149–163, 2019. doi:10.1145/3294052.3319684.

[DKM+21] Johannes Doleschal, Benny Kimelfeld, Wim Martens, Frank Neven, and Matthias Niewerth. Split-correctness in information extraction. *CoRR*, abs/1810.03367, 2021.

[DKMP22] Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:21)2022.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[Dol21] Johannes Doleschal. *Optimization and Parallelization of RegEx Based Information Extraction*. PhD thesis, University of Bayreuth and Hasselt University, 2021.

[FKP18] Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 137–149, 2018.

[FKRV13] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Spanners: a formal framework for information extraction. In *Proceedings of the 32nd Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2013. doi:10.1145/2463664.2463665.

[FKRV15a] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *Journal of the ACM*, 62(2):12:1–12:51, 2015. doi:10.1145/2699442.

[FKRV15b] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. A relational framework for information extraction. *SIGMOD Record*, 44(4):5–16, 2015. doi:10.1145/2935694.2935696.

[Fre19] Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019.

[FRU+18] Fernando Florenzano, Cristian Riveros, Martin Ugarte, Stijn Vansummeren, and Domagoj Vrgoč. Constant delay algorithms for regular document spanners. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 165–177, 2018. doi:10.1145/3196959.3196987.

[FT20] Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In *23rd International Conference on Database Theory (ICDT)*, pages 11:1–11:21, 2020. doi:10.4230/LIPIcs.ICDT.2020.11.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the 26th Symposium on Principles of Database Systems (PODS)*, pages 31–40, 2007. doi:10.1145/1265530.1265535.

[HH99] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD Conference*, pages 287–298. ACM Press, 1999.

[HMRS98] Harry B. Hunt, Madhav V. Marathe, Venkatesh Radhakrishnan, and Richard E. Stearns. The complexity of planar counting problems. *SIAM J. Comput.*, 27(4):1142–1167, August 1998.

[KLR+08] Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, Shivakumar Vaithyanathan, and Huaiyu Zhu. SystemT: A system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008.

[KNR16] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *Proceedings of the 35th Symposium on Principles of Database Systems (PODS)*, pages 13–28, 2016. doi:10.1145/2902251.2902280.

[Kre88] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490 – 509, 1988. doi:10.1016/0022-0000(88)90039-6.

[KSM95] Sampath Kannan, Z. Sweedyk, and Steve Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, SODA '95, pages 551–557. Society for Industrial and Applied Mathematics, 1995. URL: http://dl.acm.org/citation.cfm?id=313651.313803.

[Lau83] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215 – 217, 1983. doi:10.1016/0020-0190(83)90044-3.

[LBC04] Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning*, volume 3635 of *Lecture Notes in Computer Science*, pages 319–339, 2004.

[LRC11] Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, pages 109–114. ACL, 2011.

[LWYZ16] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *SIGMOD Conference*, pages 615–629. ACM, 2016.

[MRL98] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, page 426–435, 1998. doi:10.1145/276304.276342.

[MRV18] Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. Document spanners for extracting incomplete information: Expressiveness and complexity. In *Proceedings of the 37th Symposium on Principles of Database Systems (PODS)*, pages 125–136, 2018. doi:10.1145/3196959.3196968.

[MSV+19] Joshua J. Michalenko, Ameesh Shah, Abhinav Verma, Richard G. Baraniuk, Swarat Chaudhuri, and Ankit B. Patel. Representing formal languages: A comparison between finite automata and recurrent neural networks. In *ICLR (Poster)*, 2019.

[MSW16] Matús Mihalák, Rastislav Srámek, and Peter Widmayer. Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory Comput. Syst.*, 58(1):45–59, 2016. doi:10.1007/s00224-014-9571-7.

[MY18] Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In *CD-MAKE*, volume 11015 of *Lecture Notes in Computer Science*, pages 350–369, 2018.

[NBCK10] C. T. Ng, M. S. Barketau, T. C. Edwin Cheng, and Mikhail Y. Kovalyov. "product partition" and related problems of scheduling and systems reliability: Computational complexity and approximation. *Eur. J. Oper. Res.*, 207(2):601–604, 2010. doi:10.1016/j.ejor.2010.05.034.

[NG94] A. Neuwald and P. Green. Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:698–712, 1994.

[NKS+19] Galia Nordon, Gideon Koren, Varda Shalev, Benny Kimelfeld, Uri Shalit, and Kira Radinsky. Building causal graphs from medical literature and electronic medical records. In *AAAI*, pages 1102–1109. AAAI Press, 2019.

[PD07] Hoifung Poon and Pedro M. Domingos. Joint inference in information extraction. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 913–918, 2007. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-145.php.

[PFKK19] Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *Proceedings of the 38th Symposium on Principles of Database Systems (PODS)*, pages 320–334, 2019. doi:10.1145/3294052.3319699.

[PtCFK19] Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive Programs for Document Spanners. In *22nd International Conference on Database Theory (ICDT)*, pages 13:1–13:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.13.

[RBE+17] Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proc. VLDB Endow.*, 11(3):269–282, 2017. doi:10.14778/3157794.3157797.

[SC09] Robert P. Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Trans. Inf. Syst.*, 27(2):12:1–12:19, 2009. doi:10.1145/1462198.1462204.

[SM12] Charles A. Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[SWW+15] Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using DeepDive. *Proceedings of the VLDB Endowment (PVLDB)*, 8(11):1310–1321, 2015. URL: http://www.vldb.org/pvldb/vol8/p1310-shin.pdf.

[vL91] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*. MIT Press, Cambridge, MA, USA, 1991.

[WGY18] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning, (ICML)*, pages 5244–5253, 2018.

[Zac88] Stathis Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36(3):433 – 451, 1988. doi:10.1016/0022-0000(88)90037-2.

[Zuc96] David Zuckerman. On unapproximable versions of NP-complete problems. *SIAM J. Comput.*, 25(6):1293–1304, December 1996. doi:10.1137/S0097539794266407.